An architecture for non-stop upgrading of Web application

By Hoang Ha Manh

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Associate Professor Masato Suzuki

September, 2010

An architecture for non-stop upgrading of Web application

By Hoang Ha Manh (0810208)

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Associate Professor Masato Suzuki

and approved by
Associate Professor Masato Suzuki
Professor Koichiro Ochimizu
Associate Professor Toshiaki Aoki

August, 2010 (Submitted)

Abstract

Keyword: dynamic upgrading, non-stop upgrading, E-Commerce System, web application, non-stop upgrading architecture.

Today, information technology is not an unfamiliar concept. Information technology is using in many fields such as economy, transport, health care or science, research, etc… Information technology can be applied in many ways but the evident example is using software to improve business process. We can see many advantages of information technology. It can make globalization, help the company to run more effective, reduce cost of time in many activities, etc… With the important of applying information technology, software development industrial becomes an essential part in information technology.

There are many steps in the software development process: design, implementation, testing, deployment and maintenance. Maintenance step is one of the most costly steps in this process. This is especially true with web-based application software such as economic application, transfer management application or air control system, etc… These systems need to work continuously. So we need architecture for making the updating of these application systems without stopping their services.

First of all, we introduced a new way to model the web application system. For easily control the data in the system, we need a formal definition language to describe the system activities. We combine the usage of Architecture Analysis and Design Language (AADL) with a simple Sequence Diagram that we call Restricted Sequence Diagram. With this combination, we can model the data of system formally and easily to control.

After that, we determine characteristics of upgrading. Depend on the characteristics we categorize the updating type into 5 types. We find out the features of each of updating type and define each type identities.

Next, we introduced a real use case using the Electronic Commerce System. Electronic Commerce System is the standard of a commerce application so that we use it as the real example for applying our modeling method. We will find out all functions of the Electronic Commerce System and model it using the combination of AADL and Restricted Sequence Diagram.

Finally, we evaluate the ability of upgrading web application through system data model. We also discuss about the process of making non-stop upgrading without overall data affection and stopping the application services. We will have a general view of the data using in an application and determine that the non-stop upgrading can be made or not in a specific situation.

Acknowledgement

I would like to show my gratitude to all those who gave me the possibility to complete this thesis. In the first place, I would like to express my sincere gratitude and appreciation to my supervisor, Associate Professor Masato Suzuki for his constant guidance, advice, assistance and support during the whole period of my Master's course.

I would like also to express my sincere thanks to my principal supervisor Professor Koichiro Ochimizu for his encouragement and helpful comments.

I wish to say grateful thanks to Associate Professor Toshiaki Aoki and Professor Koichiro Ochimizu for their useful comments for the first draft of this dissertation and in my defense day.

I would like to thanks to Ministry of Education and Training for their financial support for my study time in Japan.

I would like to show my sincere appreciation to Japan Advanced Institute of Science and Technology for not only providing me support and great working environment but also a wonderful living condition during my study time here.

I am grateful to thanks all members in Software Structure laboratory for their kindly helps to me not only in research work but also in my daily life.

It will be a mistake if I forget to thank all members in Vietnamese group at JAIST. With all your helps, one year in here becomes enjoyable.

Last but not least, I would like to give my special thanks to my family members, especially my parents whose encouragements and helps enable me to complete my degree.

Content

List of figures

List of tables

Chapter 1:

Introduction

Nowadays, we are living in the society in which computers and the Internet is very popular. Not like twenty years ago, computers are something rare and expensive, using almost in science or industrial. At that time, programming is an unfamiliar definition in my country. Programs at that time almost are local programs, small and primary simple application running on local computer. The Internet is very expensive and very slow so that it is very hard to update a program. So there is almost no update for programs. Things are change very much now. Computer is very popular; nearly everyone has a personal computer. Now we have high-speed internet and the cost is very low. I mention these changes so we can imagine how much programs change now.

Today, software is very large and popular. Software is not only local programs but also the Internet-based programs (can be called web applications). We can see many types software such as system software, programming software, application software. The most popular type is application software. They are broadly use in many areas from entertainment to hospital or education. With the fast Internet connection, application can be updated easily. They need to be updated to fix bug, change content or design, add new features… So the update of software is one of the most important parts in development software industry.

However, when compared to a traditional application, a web application provides a wider range of application users, a wider range of tasks and interaction styles, more complex technological infrastructure and a broader range of contextual issues. For this reason, when updating a web application, we often make sequence updates instead of independent update. The updating process of web application is often called “upgrading”. So with a technique that can make the non-stop updating a component, we cannot confirm that it can be applied in the reality because we cannot sure about the data after a sequence of updating process. In a web application, almost application is very large and the data that is changed by updating a component can effect to other components. Furthermore, the data path of an application can be changed frequently because of changing business rules.

1.1. Problem

There will be no problem if all applications can use normal update method. In the simplest method, upgrades require the application to be stopped, made updates then restarted it again. This method is still acceptable with some local applications or programs that not need to work continuously.

However, as I mentioned before, the increasing of the Internet's importance and the relationship between it and global economy has made many non-stop services. Non-stop means the application run continuously without any interruption. These applications need to operate continuously because the cost of stopping service is very high. The economic cost of downtime for some online shop such as amazon.com or eBay can be hundreds of thousands of dollar per hour, the cost even higher for credit card providers, brokerages; it can be millions of dollar per hours. Besides economic applications, there are many other computer programs that must be non-stop working. This is especially necessary to critical applications such as financial applications, air traffic control systems, etc…

All of these applications need non-stop operation and still need upgrades. So the simplest update method cannot apply for these non-stop applications. So we need other method to solve this problem. Moreover, there are many other applications that do not necessarily require non-stop upgrade but would get benefit from it. For example, instead of rebooting desktop computer every time its operating system is upgraded, we would prefer to make the updates dynamically.

1.2. Objective

With above overview, we can see that the non-stop upgrade has an important role today. There are many researches to find the way to solve this problem. I will mention more about these approaches later. My research goal is to find the architecture for making non-stop upgrading web applications. However my target in the research is not focus on implementation part of making non-stop upgrading architecture. Instead of that, I focus on checking and evaluating the upgraded system to inspect about the consistency of system data.

1.3. Dissertation organization:

In chapter 2, we will discuss about the background knowledge using in this research. In this chapter, we will deal with some basic idea related to making a non-stop upgrading system. We will present about the maintenance process using in software development, about the web-based systems. We will also study about a concrete application system, the Electronic Commerce System. The last content in this chapter are some background about non-stop upgrading of application and some related work in local programming language, C for example.

In chapter 3, we will present about our approach for making non-stop upgrading for web application system. In this chapter, we will examine the tools using for modeling a system. Using these tools, we can easily control the operation of the system in component viewpoint and data viewpoint. We will use two tools in our research that are Architecture Analysis and Design Language (AADL) and Restricted Sequence Diagram. After that, we will categorize updating operation by using two mentioned tools.

The next chapter, chapter 4, we will apply our approach to analysis concrete example by using the Electronic Commerce System (E-Commerce). We will introduce about E-Commerce functions then we will model its structure by using AADL. And then, with the combination of AADL and Restricted Sequence Diagram, we will study all types of updating in E-Commerce System.

In chapter 5, we will examine about non-stop upgrading process and the step to achieve non-stop upgrading web application. And the last chapter is the conclusion and the works in the future.

Chapter 2:

Non-stop upgrading

2.1. Software maintenance

Software maintenance is a step in software development processes. Software maintenance is the modification of a software product after the deployment of that product. Software maintenance purpose is correcting bugs or improving the performance. However, software maintenance is considerably understudied area while the company and other organizations are still paying the maintenance cost. This problem is referred by Seacord et al. in 2003 and summarized by Jussi Koskinen in the following table:

Year	Proportion of software maintenance costs	Definition	Reference
2000	>90%	Software cost devoted to system maintenance & evolution / total software costs	Erlikh (2000)
1993	75%	Software maintenance / information system budget (in Fortune 1000 companies)	Eastwood (1993)
1990	>90%	Software cost devoted to system maintenance & evolution / total software costs	Moad (1990)
1990	60-70%	Software maintenance / total management information systems (MIS) operating budgets	Huff (1990)
1988	60-70%	Software maintenance / total management information systems (MIS) operating budgets	Port (1988)
1984	65-75%	Effort spent on software maintenance / total available software engineering effort.	McKee (1984)
1981	>50%	Staff time spent on maintenance / total time (in 487 organizations)	Lientz & Swanson (1981)
1979	67%	Maintenance costs / total software costs	Zelkowitz *et al.* (1979)

Table 1: Proportional software maintenance costs for its supplier.

We can see in the table, the software maintenance cost ratio is increased over time. The maintenance cost is a very large part in software development.

2.2. Web-based systems

Web-based systems are still application system but the main different is that these system work over the Internet. With this characteristic, the mechanism for making update a web-based system is different to the mechanism for making update a local system.

There are some standards for making web application system. However, in this research, we choose Java Enterprise Edition as the design standard for web application. This is 3-Tier Java Enterprise Edition application model:

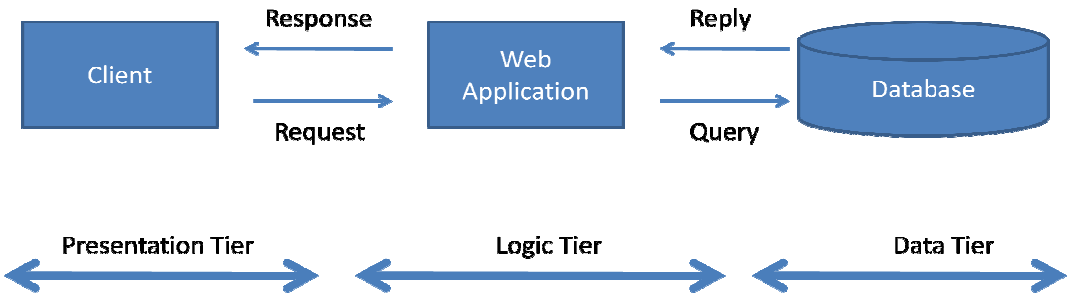


Figure 1: 3-Tiers Java Enterprise Edition application model

In this architecture, users through client send the request to web application. The request usually is the command to get some information from the system. Web application, or the business object in Java Enterprise Edition, will process these request and send the query to retrieve data to the database. The database will reply the data back to the web application and the web application will response the data to users.

2.3. The Electronic Commerce System (E-Commerce System)

The Electronic Commerce System case study is a World Wide Web-based application. E-Commerce System uses software agents as intermediaries between user interface clients and servers. In 3-Tier application model viewpoint, the agents are the business objects.

We choose the Electronic Commerce System as our concrete example for case study because of its specific features. As we know, a generic electronic commerce system is a large-scale system. So we can use it to represent to other web application system. Other feature of Electronic Commerce System is that the data in the system can change frequently because of the change of business rules. The data flow in the system is not fixed like many local systems. Each function in Electronic Commerce System can work independent to other functions.

2.4. Non-stop upgrading of application:

The maintenance cost is a large part in software development process. This is true especially with global economic system. These types of application cannot stop working in any reason or they will waste a lot of money. So it is natural to appear the need of making maintenance without stopping the system.

In order to discuss about the mechanism of non-stop upgrading, we have to distinguish the term “update” and “upgrade”. We use updating as process that we modify or replace a part of the system. “Upgrading” is close meaning to “updating”. We also use upgrading as a process that changes the system. However, upgrading is often concerned with sequence processes to make changes to the system. In other words, upgrading is the process that achieves system maintenance by applying a sequence of updating. In this dissertation, we focus on the architecture for realize the non-stop upgrading the system.

2.5. Related work:

There are some approaches to this problem. Dynamic linking is a well-known mechanism to making dynamic update. Systems based upon dynamic linking may add new code to a running program but they cannot replace existing bindings with new one (Appel 1994; Perterson et al. 1997).

In 2005, Michael Hicks and Scott Nettles presented an idea about a general-purpose framework for updating a program as it runs. They called that “dynamic software updating”. The requirement of the framework is flexible, robust, easy to use, and efficient. Their research focuses on the task of dynamically updating the code and state of a single process.

To concretize the research about “dynamic software updating”, in 2006, Iulian Neamtiu, Michael Hicks, Gareth Stoyle, and Manuel Oriol introduced a practical dynamic software updating framework for C, called Ginseng. Ginseng is an implementation of a “dynamic software updating” system.

These approaches are for C language software. For the Java application, Edward Curry, Desmond Chambers, and Gerard Lyons have designed the Chameleon framework. This framework is designed to support the use of interceptors with a Message-Oriented Middleware platform to facilitate dynamic changes.

There are some approaches for making non-stop software updating. Kuo-Feng Ssu and Hewijin Christine Jiau introduce a method to compose a program using two replicated execution blocks. By switching the execution blocks, the program can be modified without terminating its service.

Douglas Schmidt et al also introduced update patterns in 2000. There is a pattern called the Interceptor architectural pattern. This pattern allows services to be added transparently to a framework and triggered automatically when certain events occur.

However these approaches only focus on technical problem. They give us the structure to programming the non-stop software updating.

Chapter 3:

Our approach for non-stop upgrading

3.1.Formal definition of an application

In order to archive the goal, we need tools that can help us to describe the system formally. We need a language to model the application precisely but not so particularly. We only focus on the data and how the system controls the data, so we choose Architecture Analysis and Design Language in this research.

3.1.1. Architecture Analysis and Design Language (AADL)

The AADL is a modeling language released by the Society of Automotive Engineers in November 2004. The AADL provides us formal modeling concepts; we can use these concepts to describe and analyze applications or systems architecture. Architecture Analysis and Design Language describe the system in terms of components and their interactions so that AADL is effective for model-based analysis of system applications.

The central element in AADL is component. Each component will be assigned a unique identity. A component is declared by type and implementation. Component type is used for define interface elements and external attributes of the component. Otherwise, component implementation declaration defines the component's internal structure. There are three sets of components in AADL:

- Application software: thread, thread group, process, data, subprograms.
- Execution platform: processor, memory, device, bus.
- Composite: combine of two such as system.

The AADL structure for defining a component type is as following:

```
component_type {name}
extends {component type}
features
flows
properties
```

A component {name} of type {component_type} can be extended by another component type. {features} are the interfaces of this component. {flows} specify channels of information transfer in this component. {properties} define intrinsic characteristics of the component.

A component implementation specifies an internal structure of the component.

```
Type implementation {type name}.{implementation name}
     extends
     refines type
     subcomponents
     calls
     connections
     flows
     modes
     properties
```

Besides two main elements component type and component implementation, AADL language still has other elements. There are packages, property sets and annexes. However, in this research, we only care about component type and component implementation so we do not need to go to detail with other elements.

An AADL specification can be expressed by three types: text, graphic or Extensible Markup Language (XML).

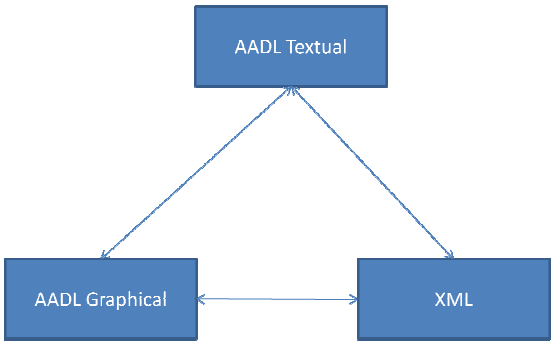


Figure 2: 3 types of an AADL specification

3.1.2. Restricted Sequence Diagram

We can use AADL to describe the elements of system. Using AADL we can easily control the components and the transference of data in the system. We can easily see how many data port each component has, where the data goes. However, we cannot determine the order of data flows so we use another tool for solving this problem. A Sequence Diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with other processes and their order. In our research, we do not need to use all function of UML Sequence Diagram, we use some main idea of Sequence Diagram to help us in describe the system more clearly. We call this Restricted Sequence Diagram.

The reasons that we call it Restricted Sequence Diagram because of two following reasons:

- The first one is that we will ignore guard conditions of the data flow in original Sequence Diagram. In Sequence Diagram, some flows need the specific condition with the format *[Condition] f*. However, in Restricted Sequence Diagram, we ignore this.
- The second one is that we will ignore the arguments of the data flow because these arguments are described in AADL.

In Restricted Sequence Diagram, we care about three elements: objects, flows and their order. For each of flow *f*, we have the source of *f* and destination of *f*. The source and the destination of a flow is belongs to the set of objects. The order of flows is a reflex from natural number to the flow *f*.

Definition 3.1:

Restricted Sequence Diagram can be defined as following formula:

$RSD = (O, F, order)$

$where$:

O: $set\ of\ objects\ or\ instances$

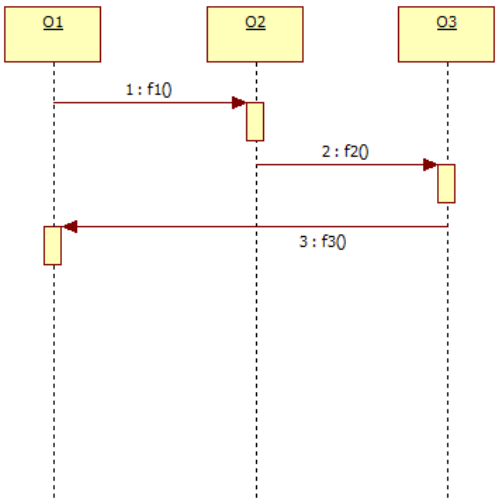
F: $set\ of\ message\ flows$

$\forall f \in F: f(source, destination)$

$source, destination \in O$

order: $\mathbb{N} \rightarrow \mathrm{F} = \{< 1, \mathrm{f}_1 >, < 2, \mathrm{f}_2 >, \dots, < k, \mathrm{f}_\mathrm{k} >, \dots\}$

Figure 3 is an example of Restricted Sequence Diagram. In this example, we have 3 object o1, o2, o3; three flow f1 from o1 to o2, f2 from o2 to o3, f3 from o3 to o1. And the order of three flow is f1, f2, f3 corresponding.



$In\ this$:

$O = \{o_1, o_2, o_3\}$

$F = \{f_1\ (o_1, o_2), f_2(o_2, o_3), f_3(o_3, o_1)\}$

$order$: $\{< 1, f_1 >< 2, f_2 >< 3, f_3 >\}$

Figure 3: An example of Restricted Sequence Diagram

Using Restricted Sequence Diagram, we can express the system's action with more accurately and more precisely. Combining with AADL, we can easily describe dependencies caused by message flows of the system.

3.2. Categorization of updating types

We know that a web application is a large-scale system. So the update also has many different forms. In order to research about non-stop upgrading, we need to distinguish these kinds of updating to know about the different of data using in each type and the identity of each updating type.

In our research, we categorize updating type of system into 5 types. The purpose of this is specifying each type of updating

- Type 0: the component is updated only inside of this component.
- Type 1: the component will have a new connection to a new component.
- Type 2: the component will have a new connection with existed component and the direction of this connection is not specified.
- Type 3: the component will have a new connection with existed component like type 2 but the direction of this connection is depend on existed connection.
- Type 4: a new component will be added between two existed components.

In this part, we use some definition as following:

$\Delta O: Set\ of\ updated\ objects\ in\ Restricted\ Sequence\ Diagram.$

$\Delta F: Set\ of\ updated\ flows\ in\ Restricted\ Sequence\ Diagram.$

$ports(o): The\ set\ of\ data\ ports\ of\ o\ where\ o\ is\ a\ component.$

$used(f): The\ set\ of\ data\ ports\ used\ by\ f\ where\ f\ is\ a\ data\ flow.$

Where *ports* and *used* are calculated from AADL description corresponding to component *o* and flow *f*.

The compatible between two components is defined as the port of first component o1 is equivalent to the port of second component o2. That means component o1 can be replaced by the component o2 without changing of message flow or data flow and vice versa.

Definition 3.2:

$We\ define\ the\ concept\ about\ the\ compatible\ between\ two\ components$:

$compatible(o_1, o_2) \equiv ports(o_1) \sim ports(o_2)$

3.2.1. Type 0:

We call first type of updating as updating type 0. In this type, the updated component only has internal changes. This updating type is implemented a lot in reality because we need to change function of a method frequently. For example, we often need some minor update to application such as change the message to notify the customer or change the business of a function.

In Figure 4, the updated version C1' of C1 is made by updating C1's internal elements, F1 and F2.

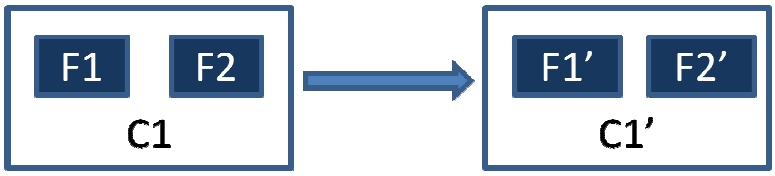


Figure 4: Updating Type: Type 0

In this type, the input and output data flow of new component is same to old component. All of new component data ports must be same to old component data ports. In other words, this type of update does not change the component from outside viewpoint. So the new component F1' must be an extension of old component F1. Similarly, F2' must be an extension of old component F2.

Definition 3.3:

$We\ assume\ that\ the\ system\ S\ before\ updating\ is$

$S = \{C_1\}\ with\ C_1 = (F_1, F_2)$

$so\ the\ system\ after\ updating\ is$

$S' = \{C_1'\}\ with\ C_1' = (F_1', F_2')$

$Updating\ type\ 0\ of\ system\ S\ is$:

$U_0\ [S] =\ S'$

$where$

$compatible\ (F_1\ , F_1'\ )\ and\ compatible\ (F_2\ , F_2'\ )$

The compatible between two internal parts of a component means that the updated part does not change the function and data flow of this component.

Characteristic of type 0 updating:

- AADL:
    - Number of components is unchanged.
    - Data port: same
    - Data flow: same
- Restricted Sequence Diagram
    - Set of object: same
    - Set of flow: same

3.2.2. Type 1:

The next type of updating is called updating type 1. In this type, a component in system will have a new connection to a new component. For example, instead of getting only customer information, we want to update the customer business object to get the total fund that the customer spent. In this case, the customer business needs to have a new connection to account database, for example, to get the extra data.

Figure 5 is showed the description model that the component P will have new connection to new component C_{new}.

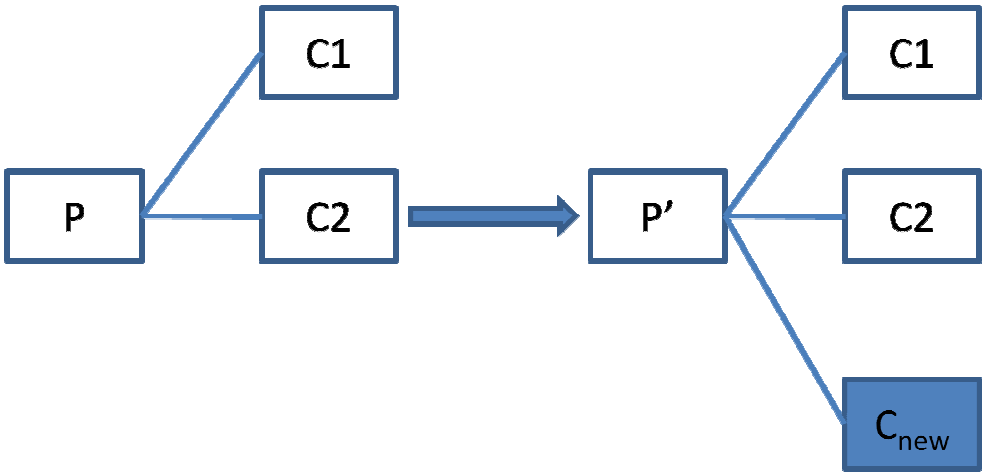


Figure 5: Updating Type: Type 1

Definition 3.4:

$We\ assume\ that\ the\ system\ S\ before\ updating\ is$:

$S\ =\ \{P, C_1, C_2\}$

$So\ the\ system\ S'\ after\ updating\ will\ be$:

$S' =\ \{P', C_1, C_2, C_{new}\}$

$Updating\ type\ 1\ of\ system\ S\ is$:

$U_1\ [S]\ =\ (S')$

Characteristic of type 1 updating:

- AADL:
  - Number of components after update increased by one.
  - Data port: data ports of P are increased proportional to the addition of the new component.
  - Data flow: similar to the changing of data ports.
- Restricted Sequence Diagram:
  - Set of object: set O' (after update) is a superset of set O.

    $O'\ =\ O \cup\ \Delta O$
  - Set of flow: similarly, set F' (after update) is a superset of set F.

    $F' = F \cup\ \Delta F$
  - New flows must be related only to new objects.

    $used(\Delta F) = ports(\Delta O) \cup \left(ports(P') \setminus ports(P)\right)$

3.2.3. Type 2:

Next, the updating type 2 has a little similar to the updating type 1. In type 2 updating, a component will have a new connection with an existed component instead of a new component. The direction of this connection is not considered.

For example, the original version of a function is getting id and price of products. If we update the component to get the extra description from the same database, this is called the updating type 2.

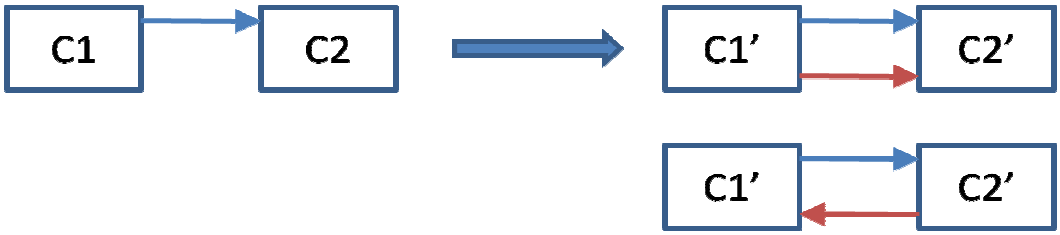


Figure 6: Updating Type: Type 2

Consider this updating model in Figure 6, after updating we have a new connection from C_1' to C_2' (in red color). The order of this connection can be from C_1' to C_2' or vice versa.

Definition 3.5:

$Updating\ type\ 2\ of\ two\ components\ C1\ and\ C2\ is$:

$U2\ [S]\ =\ (S')$

$where$

$\begin{cases} S\ =\ (C_1, C_2) \\ S'\ =\ (C_1', C_2') \end{cases}$

$and\ ports(C_1) \subset ports(C_1')$

$and\ ports(C_2) \subset ports(C_2')$

Characteristic of type 2 updating:

- AADL:
  - Number of components is unchanged.
  - Data port: data ports after updating are increased corresponding to the flow.
  - Data flow: data flow after updating increased by one.
- Restricted Sequence Diagram:

- Set of object: set O' is same to set O.
- Set of flow: set F' (after update) is a superset of set F.

3.2.4. Type 3:

Updating type 3 is similar to updating type 2 except the direction of new data flow is depend on previous data flow. For example, after buying items, the application will store the order in the order database. However; if we want more guarantee, we will update the application that response the confirmation for successful updating.

Figure 7: Updating Type: Type 3

The new data flow (in red color) depends on the existed data flow.

Definition 3.6:

$Updating\ type\ 3\ of\ two\ components\ C_1 and\ C_2\ is$

$U_3[S] = (S')$

$where$

$\begin{cases} S = (C_1, C_2) \\ S' = (C_1', C_2') \end{cases}$

$and\ ports(C_1) \subset ports(C_1')$

$and\ ports(C_2) \subset ports(C_2')$

Characteristic of type 3 updating:

- AADL:
  - Number of components is unchanged.
  - Data port: data ports after updating are increased corresponding to flow.
  - Data flow: data flow after updating increased by one.
- Restricted Sequence Diagram:

- Set of object: set O' is same to set O.
- Set of flow: set F' (after update) is a superset of set F.

  $We\ assume\ that\ order\ of\ f \in \Delta F\ is\ n\ and\ f\ has\ form\ f(o_{source}, o_{destination})$
  $then\ f_{n-1}\ at\ order\ n-1\ must\ have\ form\ f_{n-1}(o_{destination}, o_{source})$
- Order: in this type, the order will have specific condition: the source of new data flow must be the destination of existed data flow.

3.2.5. Type 4:

In this final type of updating, there will be a new component added between two existed components. Data flow between two existed components will be sent to a middle component, the new one.

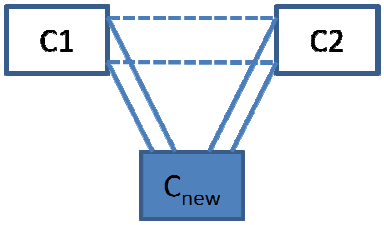


Figure 8: Updating Type: Type 4

In the example, data between two existed components, C1 and C2, will pass to an intermediate component, a new component, C_{new}.

Definition 3.7

$U_4[S] = (S')$
$where\ S = (C_1, C_2)\ and\ S' = (C_1, C_2, C_{new})$
$and\ ports(C_{new}) = ports(C_1) \cup ports(C_2)$

Characteristic of type 4 updating:

- AADL:
  - Number of components is increased by one.
  - Data port: data ports after updating is more than those before updating.

    $\forall p \in ports(\Delta F),\ p \in ports(C_{new}) \cup ports(C_1) \cup ports(C_2)$
  - Data flow: data flow after updating increased too.
- Restricted Sequence Diagram:

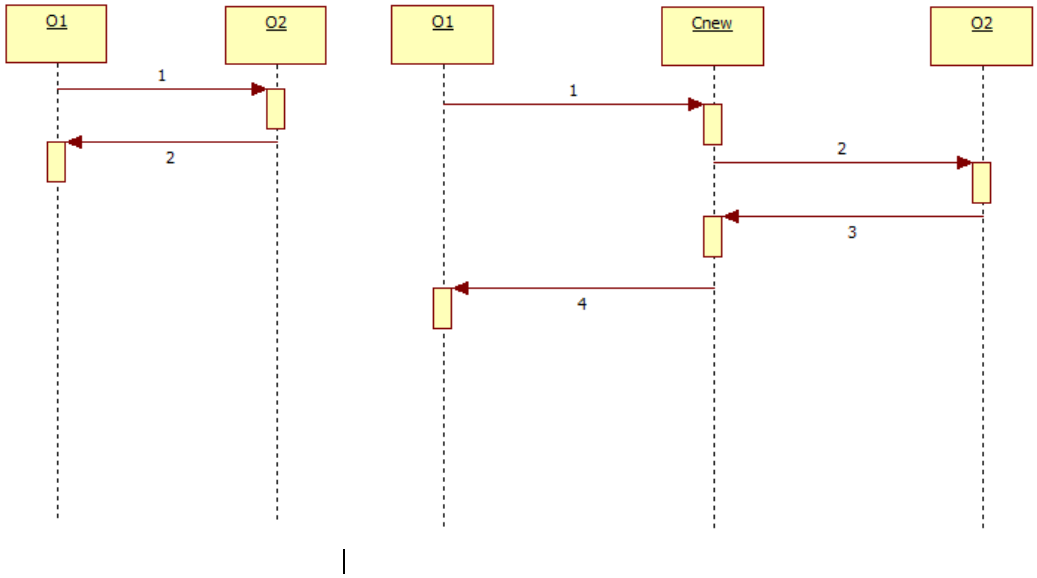


Figure 9: Restricted Sequence Diagram of updating type 4

  - Set of object: number element of set O' is more than the number element of set O.

    $O' = O \cup \Delta O$
  - Set of flow: set F' (after update) is

    $F' = \Delta F$

3.3. Our architecture for upgrading:

3.3.1. Interceptor pattern:

The Interceptor architectural pattern is one of pattern-oriented software architecture introduced in as a pattern for concurrent and networked objects. The Interceptor architectural pattern allows services to be added transparently to a framework and triggered automatically when certain events occur.

So we can use this architecture to control data flow in the system.

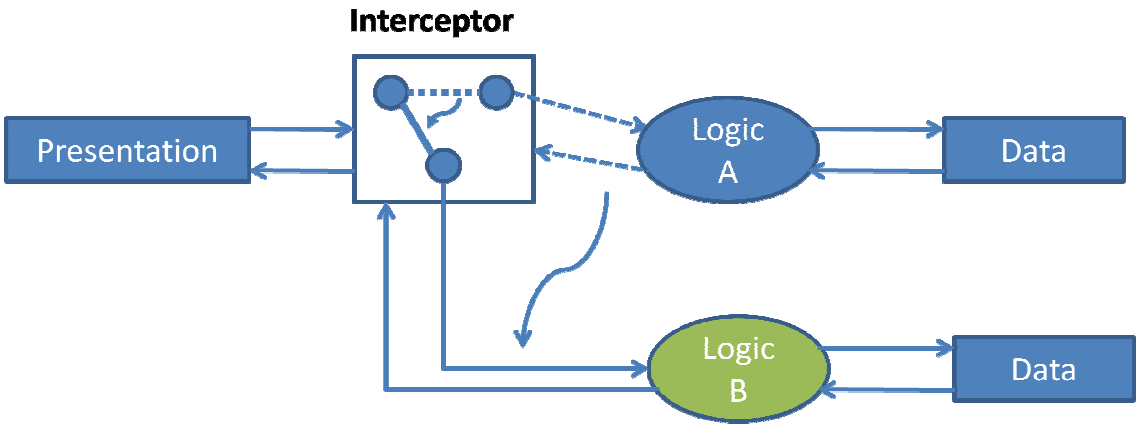


Figure 10: Concept diagram of Interceptor architecture

In our research, we define architecture depend on the idea of the Interceptor architectural pattern.

3.3.2. Our defined architecture:

We want to improve the behavior of Interceptor pattern by introducing Interceptor Configuration. The Interceptor Configuration will control the activities of Interceptor to perform the updating in the system.

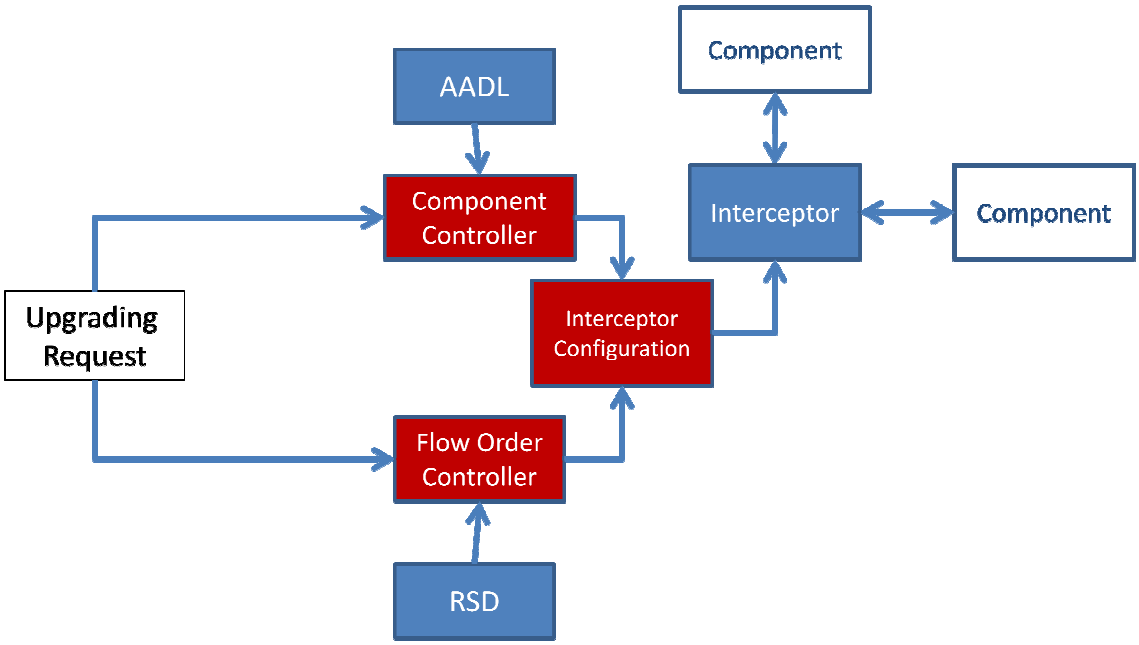


Figure 11: Our architecture for non-stop upgrading

In our architecture, after sending upgrading request to the system, the request will be sent to two controllers: Component Controller and Flow Order Controller. These two controllers will affect the Interceptor Configuration. The Interceptor Configuration will control the behavior of Interceptor in the system so that it will control the data flow in the system to perform non-stop upgrading.

3.3.2.1. Role of Component Controller

The Component Controller will get the information about the changing of components in the system depend on AADL of system before and after each updating.

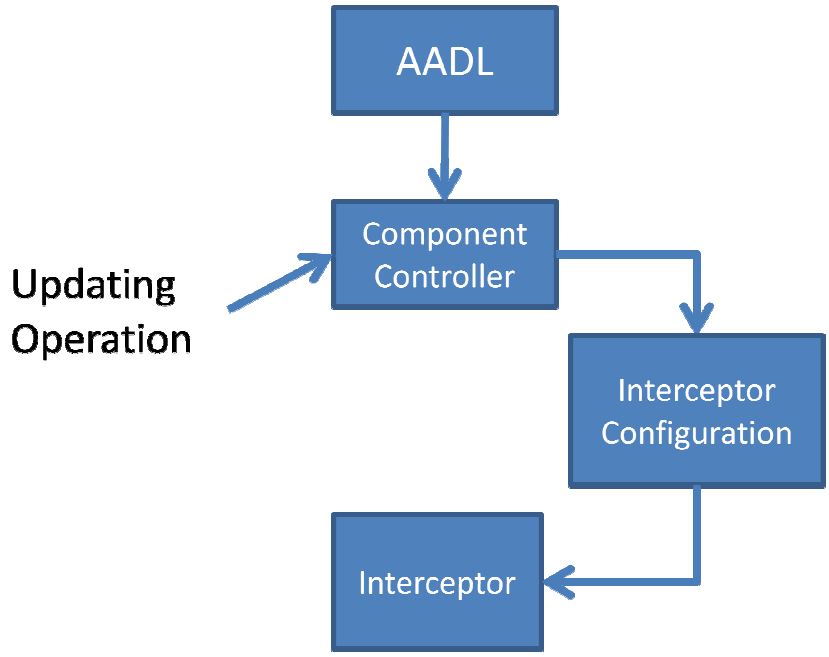


Figure 12: Component Controller

The Component Controller has these functions:

- Manage components and data ports before and after update
- Send this information to Interceptor Configuration to determine what update will be performed.

3.3.2.2. Role of Flow Order Controller

The Flow Order Controller will get information from RSD specification and send information to Interceptor Configuration to control the updating time of the system.

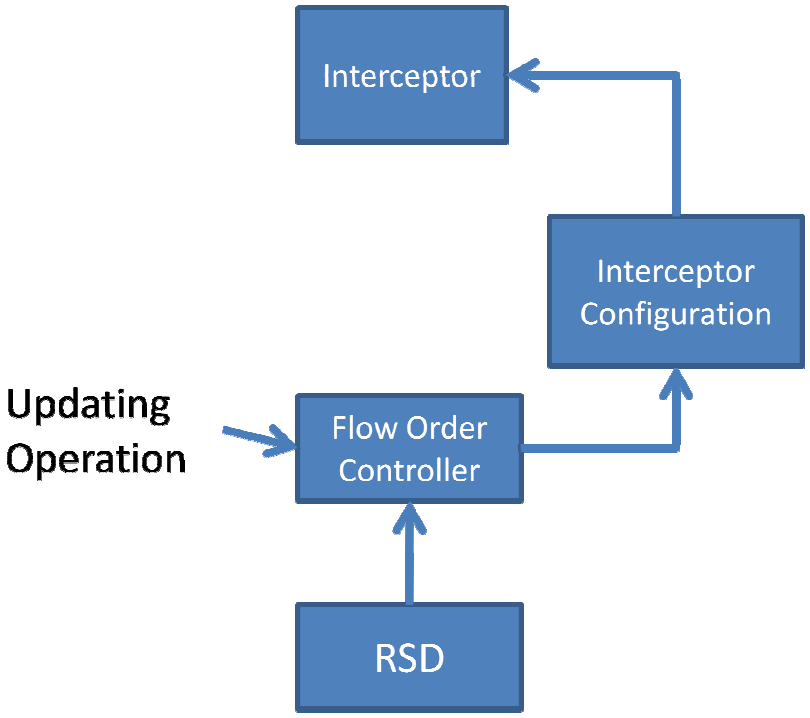


Figure 13: The Flow Order Controller

The role of Flow Order Controller:

- Manage the order of message flow before and after update
- Send information to Interceptor Configuration to determine the suitable time to update.

3.4. Our updating architecture specification:

3.4.1. Type 1

3.4.1.1. Specification and updating operation request

In updating type 1, we have two components as following:

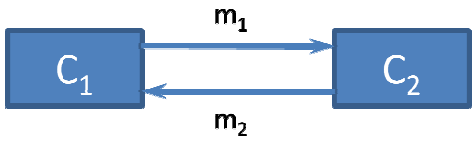


Figure 14: Updating type 1 - Before

After updating, the diagram changes into:

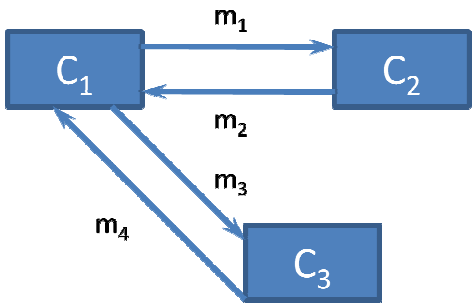


Figure 15: Updating type 1 - After

Update Operation request will be sent to the controllers with following specification:

```
Update_Type1 (C1)
{
      Add_Component (C3);
      Add_Connection ( m3 (C1, C3) );
      Add_Connection ( m4 (C3, C1) );
}
```

Upgrading is a sequence of updating so that the specification included the updating type and the component being updated:

```
Update_Type1 (C1)
```

In this case, updating type 1 will have 3 commands. The first one is the command that add a new component. Two next commands are two adding connection commands. The connection adding command will include the name of the new connection, the source and the destination of the connection.

```
Add_Component (C3);
Add_Connection ( m3 (C1, C3) );
Add_Connection ( m4 (C3, C1) );
```

3.4.1.2. The real architecture using Interceptor Proxy

In the real implementation of our architecture, Interceptor pattern has the role of intermediate controller between components in the system. Interceptor will control data flow in the system in order to manage system updating.

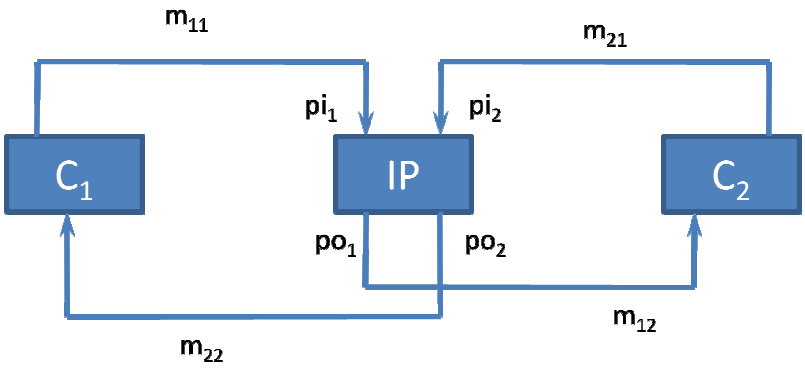


Figure 16: Interceptor implementation in updating type 1

Interceptor Configuration store the information about activities of Interceptor Proxy

```
m1 (C1, C2)
{
      m11 (C1, IP: pi1)
      m12 (IP: po1, C2)
}
m2 (C2, C1)
{
      m21 (C2, IP: pi2)
```

```
        m22 (IP: po2, C1)
}
```

The first line define the name of connection including the source and destination of it

```
m1 (C1, C2)
```

This connection will be managed by two real connections. In each of these connections, the specification describes the source and the destination. If the destination or source is Interceptor, we will specify the input or output port of Interceptor.

```
        m11 (C1, IP: pi1)
        m12 (IP: po1, C2)
```

For example, the connection m_{11} from C_1 to Interceptor will be specified the input port pi_1 in the Interceptor.

The implement system after updating is described as following:

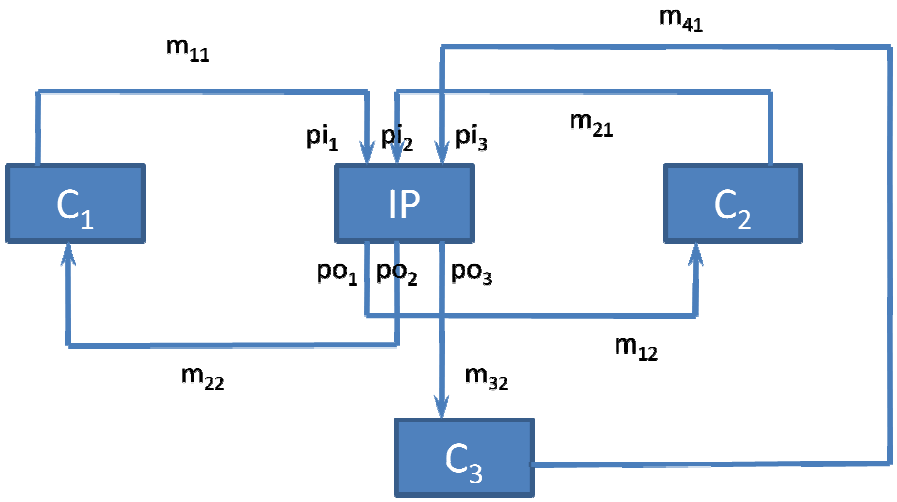


Figure 17: Interceptor implementation after updating type 1

```
m1 (C1, C2)
{
      m11 (C1, IP: pi1)
      m12 (IP: po1, C2)
}
m2 (C2, C1)
{
      m21 (C2, IP: pi2)
      m22 (IP: po2, C1)
}
// New elements:
m3 (C1, C3)
{
      m11 (C1, IP: pi1)
      m32 (IP: po3, C3)
}
m4 (C3, C1)
{
      m41 (C3, IP: pi3)
      m22 (IP: po2, C1)
}
```

Depend on the specification of Interceptor Configuration; we can determine the updating time when m_{22} is finished.

3.4.2. Type 2

3.4.2.1. Specification and updating operation request

In updating type 2, we have two components as following

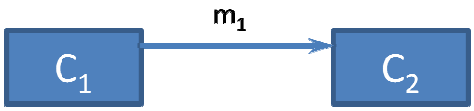


Figure 18: Updating type 2 - Before

After updating, the diagram changes into:

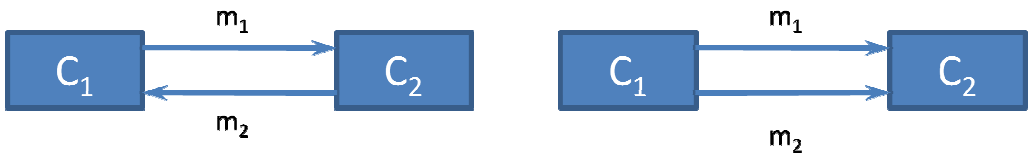


Figure 19: Updating type 2 - After

Update Operation request will be sent to the controllers with following specification for each case:

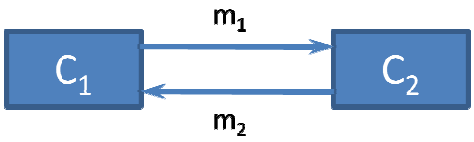


Figure 20: Case 1

```
Update_Type2 (C1,C2)
{
      Add_Connection ( m2 (C2, C1) );
}
```

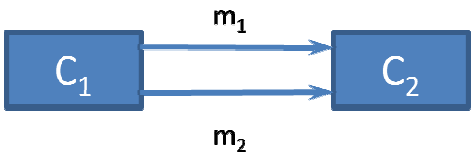


Figure 21: Case 2

```
Update_Type2 (C1,C2)
{
      Add_Connection ( m2 (C1, C2) );
}
```

3.4.2.2. The real architecture using Interceptor Proxy

The implementation using Interceptor before updating is as following:

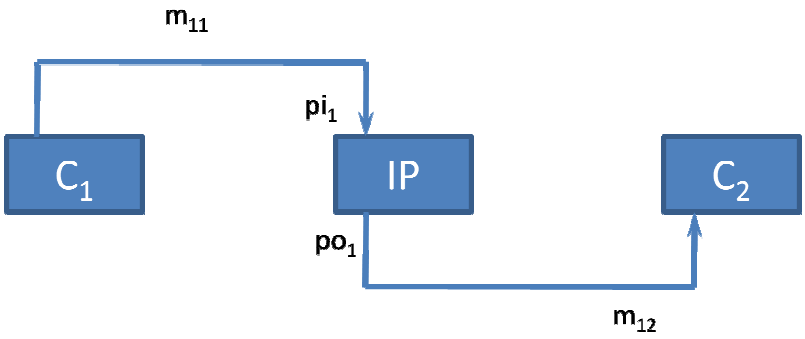


Figure 22: Interceptor in type 2 - Before

```
m1 (C1, C2)
{
        m11 (C1, IP: pi1)
        m12 (IP: po1, C2)
}
```

And after updating:

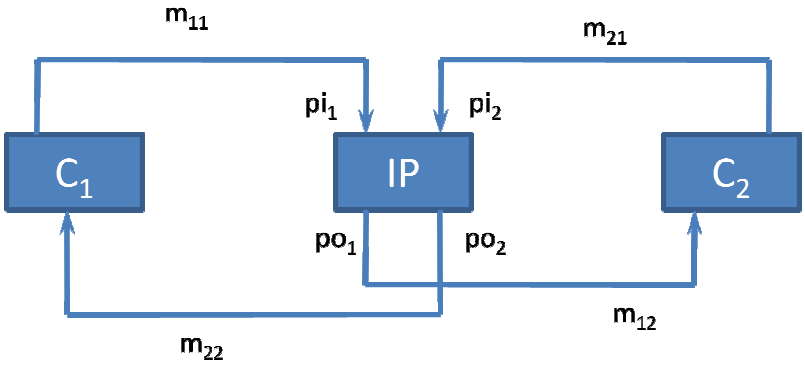


Figure 23: Interceptor in type 2 – After (Case 1)

```
m1 (C1, C2)
{
	m11 (C1, IP: pi1)
	m12 (IP: po1, C2)
}
//New
m2 (C2, C1)
{
	m21 (C2, IP: pi2)
	m22 (IP: po2, C1)
}
```

Case 2 of updating type 2:

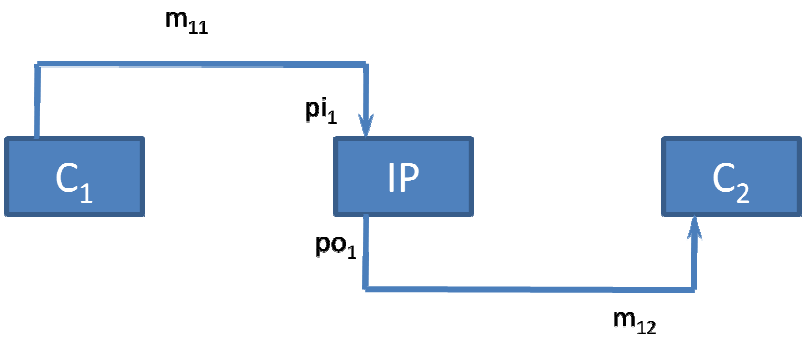


Figure 24: Interceptor in type 2 – After (Case 2)

```
m1 (C1, C2)
{
	m11 (C1, IP: pi1)
	m12 (IP: po1, C2)
}
//New
m2 (C1, C2)
{
	m11 (C1, IP: pi1)
	m12 (IP: po1, C2)
}
```

3.4.3. Type 3

3.4.3.1. Specification and updating operation request

In updating type 3, we have two components as following before updating:

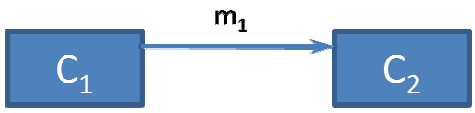


Figure 25: Updating type 3 - Before

After updating, the diagram changes into:

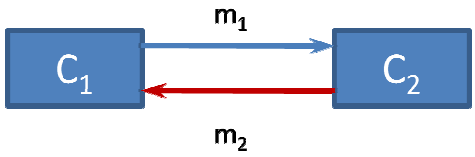


Figure 26: Updating type 3 - After

Update Operation request will be sent to the controllers with following specification:

```
Update_Type2 (C1,C2)
{
      Add_Connection ( m2 (C2, C1) );
}
```

3.4.3.2. The real architecture using Interceptor Proxy

The implementation using Interceptor before updating is as following:

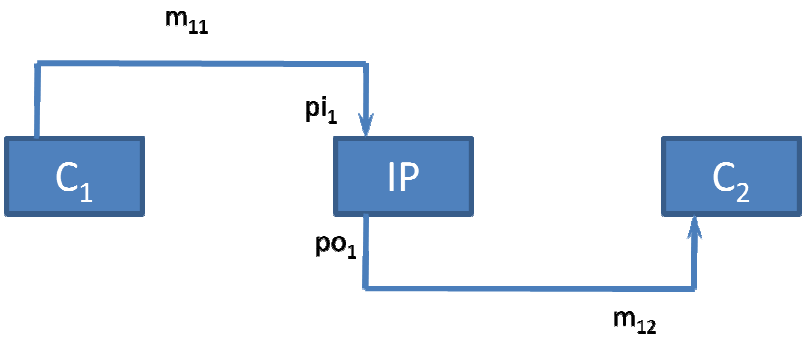


Figure 27: Interceptor in type 3 - Before

```
m1 (C1, C2)
{
        m11 (C1, IP: pi1)
        m12 (IP: po1, C2)
}
```

After Updating

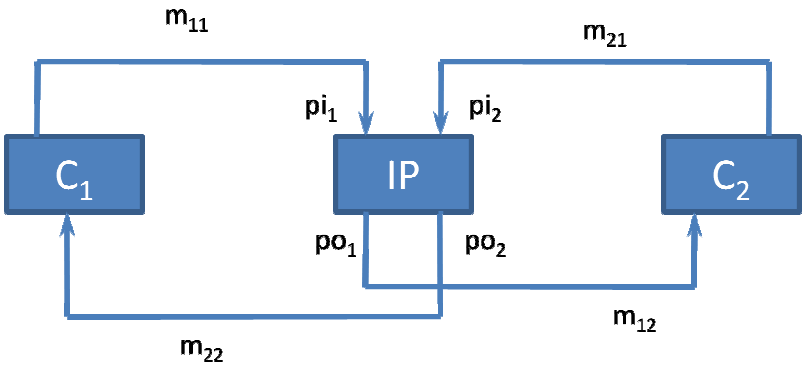


Figure 28: Interceptor in type 3 – After

```
m1 (C1, C2)
{
      m11 (C1, IP: pi1)
      m12 (IP: po1, C2)
}
m2 (C2, C1)
{
      m21 (C2, IP: pi2)
      m22 (IP: po2, C1)
}
```

3.4.4. Type 4

3.4.4.1. Specification and updating operation request

In updating type 4, we have two components as following

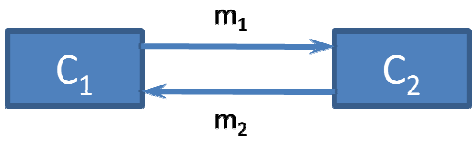


Figure 29: Updating type 4 - Before

After updating, the diagram changes into:

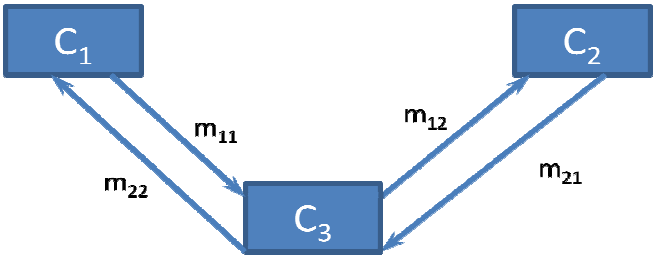


Figure 30: Updating type 4 - After

Update Operation request will be sent to the controllers with following specification:

```
Update_Type4 (C1, C2)
{
      Add_Component (C3);
      Add_Connection ( m11 (C1, C3) );
      Add_Connection ( m12 (C3, C2) );
      Add_Connection ( m21 (C2, C3) );
      Add_Connection ( m22 (C3, C1) );
}
```

3.4.4.2. The real architecture using Interceptor Proxy

The implementation using Interceptor before updating is as following:

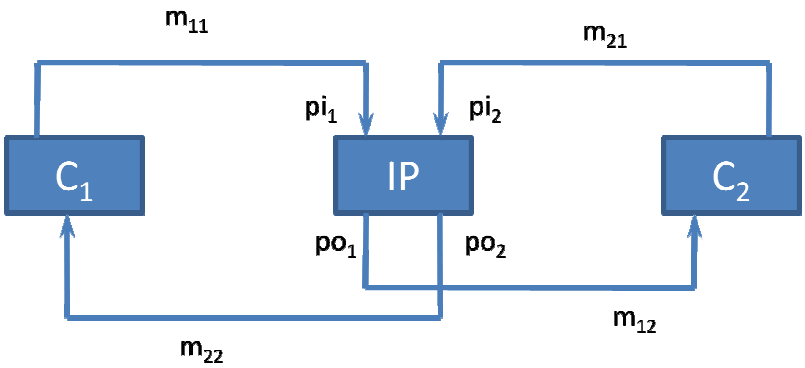


Figure 31: Interceptor in type 4 - Before

Interceptor Configuration stores the information about activities of Interceptor:

```
m1 (C1, C2)
{
      m11 (C1, IP: pi1)
      m12 (IP: po1, C2)
}
m2 (C2, C1)
{
      m21 (C2, IP: pi2)
      m22 (IP: po2, C1)
}
```

After Updating:

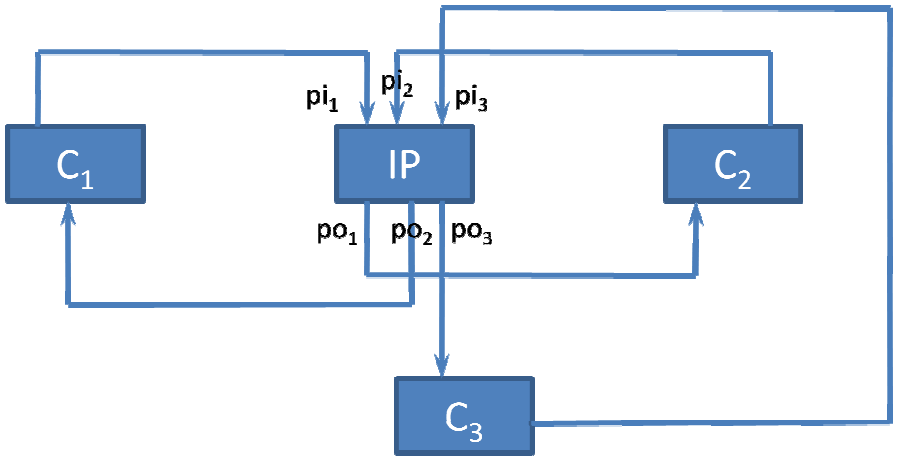


Figure 32: Interceptor in type 4 - After

```
m11 (C1, C3)
{
      m111 (C1, IP: pi1)
      m112 (IP: po3, C3)
}
m12 (C3, C1)
{
      m121 (C3, IP: pi3)
      m122 (IP: po2, C1)
}
m21 (C2, C3)
{
      m211 (C2, IP: pi2)
      m212 (IP: po3, C3)
}
m22 (C3, C1)
{
      m221 (C3, IP: pi3)
      m222 (IP: po2, C1)
}
```

Chapter 4:

Case study

After preparing background knowledge about the using of Architecture Analysis and Design Language and Restricted Sequence Diagram, categorizing types of updating, we have the tools to describe the system. In this chapter, we will put them into practice by apply to the concrete example. In this case, we will analyze the updating types on the Electronic Commerce System case study (E-Commerce).

4.1. Overview functions of E-Commerce System

In the electronic commerce system, there are customers and suppliers. Each customer has a contract with a supplier for purchases from that supplier and has one or more bank accounts to make the payments to suppliers. Each supplier provides a catalog of items, accepts customer orders, and receives payment from customers.

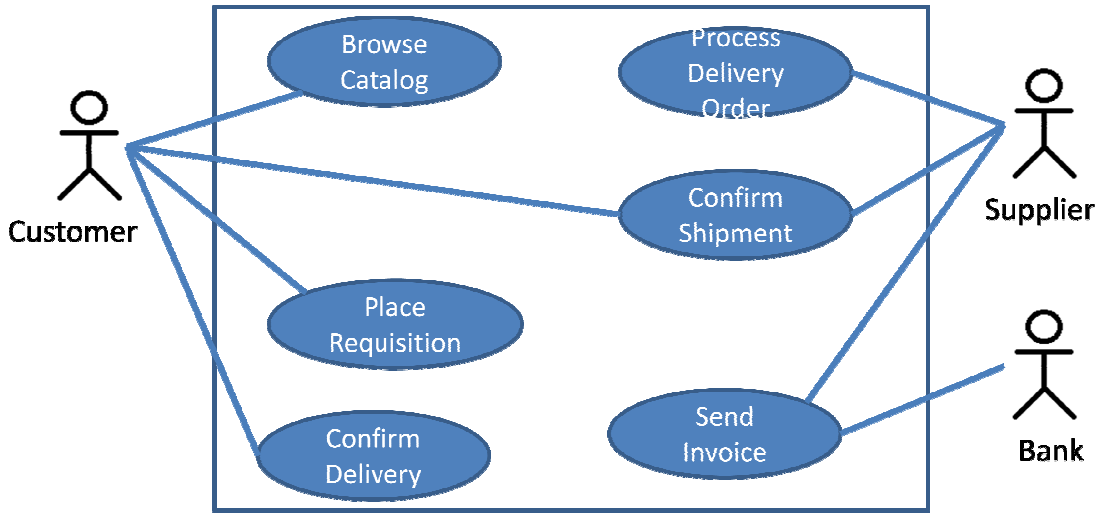


Figure 33: E-Commerce Function

A customer is able to browse several catalogs provided by the suppliers and make selections of items that he need to purchase. The customer's order needs to be checked against the available contracts to determine if there is a valid customer contract with the supplier. These contracts will be used for charging the purchase. Each contract has operations funds committed to it. It is necessary to determine that sufficient funds are available for the customer order. Assuming that the contract and funds are in place, a delivery order is created and sent to the catalog supplier. The supplier confirms the order and enters a planned shipping date. When the order is shipped, the customer is notified. The customer acknowledges when the shipment is received and the delivery order is updated. After receipt of shipment, authorization for payment of the invoice is made. The invoice is checked against the contract, available funds, and delivery order status. After that, the invoice is sent to accounts payable which authorizes payment of funds. Payment is made through electronic funds transfer from the customer bank to the supplier bank.

The data using in E-Commerce is list in the following table:

Entity	Data
Customer	customerID: Integer address: String telephoneNumber: String faxNumber: String
Inventory	itemID: Integer itemDescription: String quantity: Integer price: Real reorderTime: Date
BankAccount	bankID: Integer locationOfBank: String bankAccountNumber: String accountType: String
DeliveryOrder	orderID: Integer plannedShipDate: Date actualShipDate: Date

	creationDate: Date orderStatus: String amountDue: Real receivedDate: Date
Contract	contractID: Integer maxPurchase: Real
Supplier	supplierID: Integer address: String telephoneNumber: String faxNumber: String
Invoice	invoiceID: Integer amountDue: Real invoiceDate: Date
SelectedItem	itemID: Integer unitCost: Real quantity: Integer
Catalog	itemID: Integer itemDescription: String unitCost: Real
Payment	paymentID: String amount: Real date: Date status: String
Requisition	requisitionID: Integer amount: Real status: String
OperationFunds	operationFundsID: Integer totalFunds: Real committedFunds: Real reservedFunds: Real

Table 2: Structure of all data entity in E-Commerce System

In E-Commerce System, there are six functions.

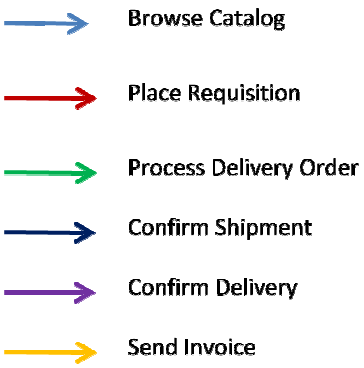


Figure 34: Six function of E-Commerce System

4.1.1. Browse Catalog:

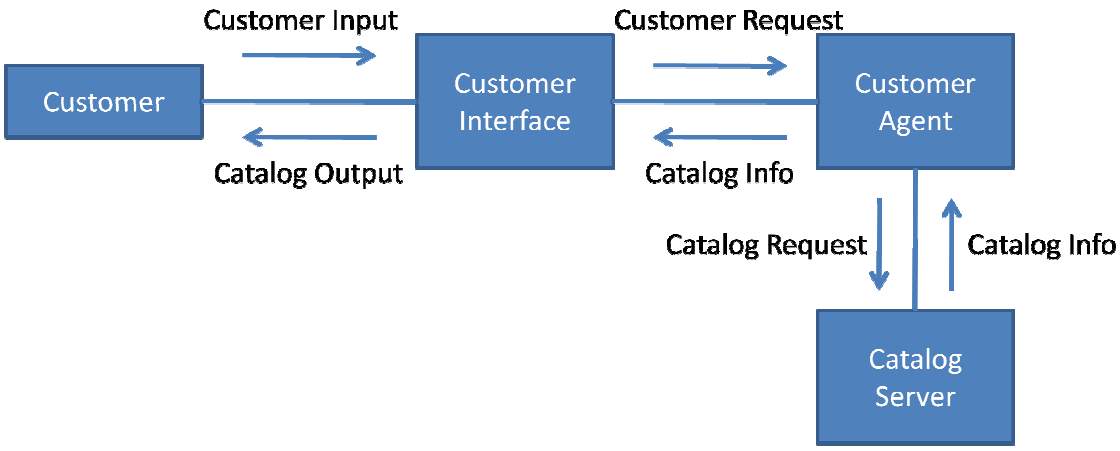


Figure 35: Browse Catalog Function

First function is Browse Catalog function. Using this function, users use Customer Client to send a browse catalog request. After receiving the request from client, customer business object will make a query to server to get the catalog information from catalog database. After that, customer business object will response the data back to customer client.

4.1.2. Place Requisition:

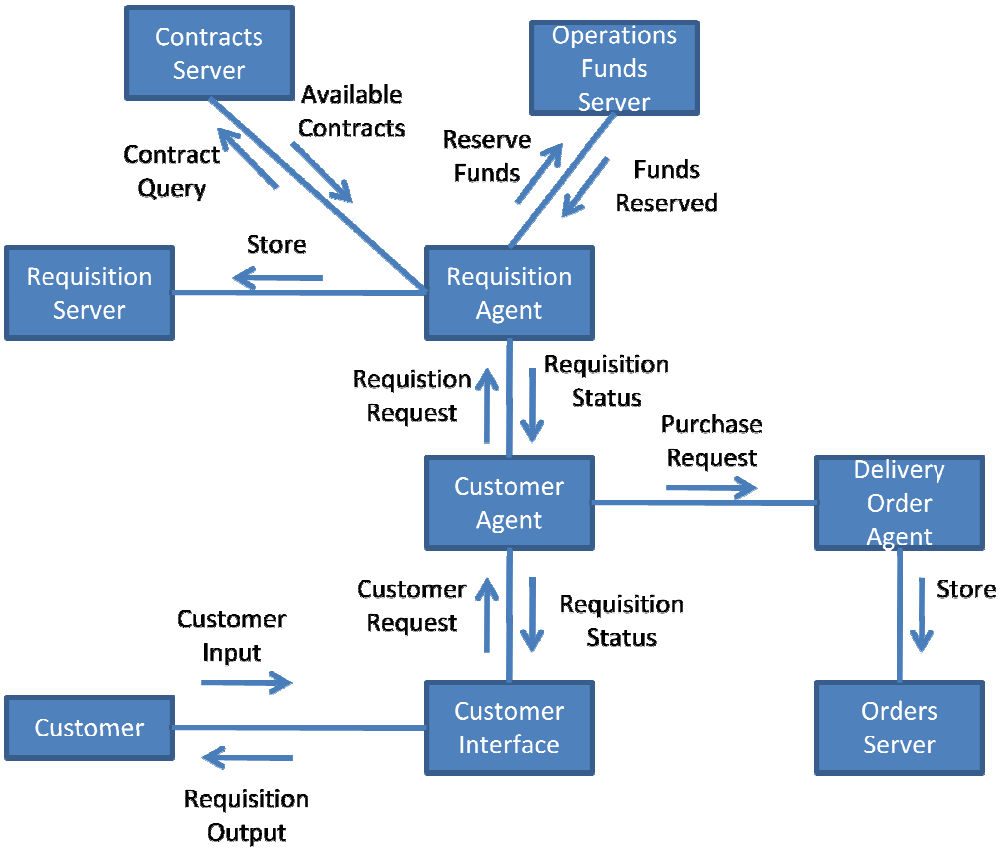


Figure 36: Place Requisition Function

The next function in E-Commerce System is Place Requisition function. In this function, a customer will select item from catalog and send the request to create a requisition through Customer Client. Then the request will be sent to Customer Business Object. Customer Business Object instantiates the Requisition Business Object and passes customer's request to it.

Next, Requisition Business Object will check from Contracts Database to check whether the contract between the customer and the supplier is existed. If contract is ok, Requisition Business Object will send a reserve funds request to the Operations Funds Server to hold the funds from a given contract for this requisition.

After receiving confirm that the funds have been reserved, Requisition Business Object approves the requisition and stores at the Requisition Database. Requisition Business Object sends the requisition status to the Customer Business Object then the Customer Business Object instantiates a Delivery

Order Business Object and sends the purchase request to it. Delivery Order Business Object stores new delivery order in Orders Server.

Finally, Customer Business Object sends the requisition status to the Customer Client to display the status to the customer.

4.1.3. Process Delivery Order:

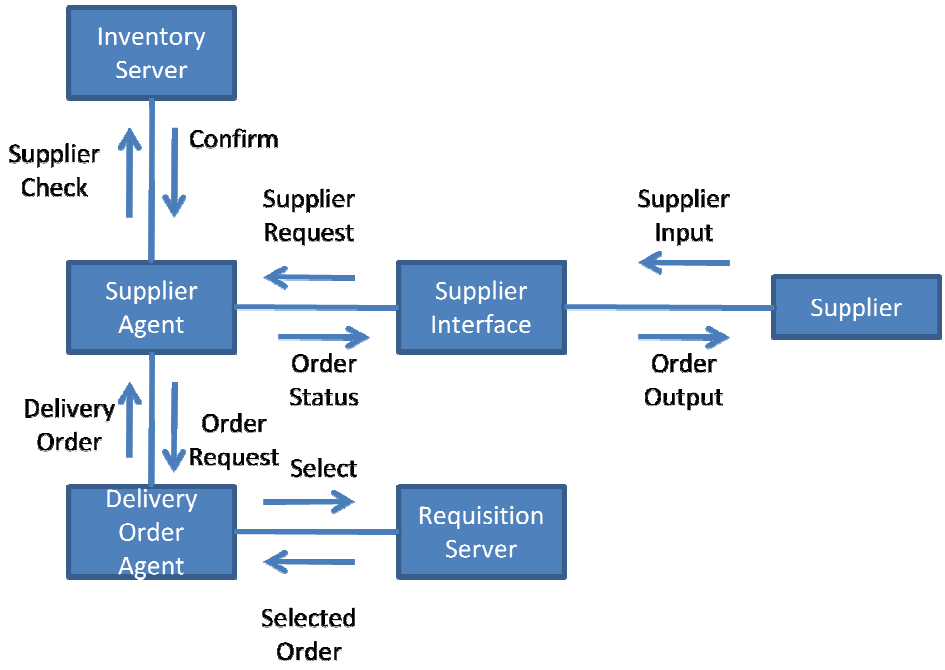


Figure 37: Process Delivery Order Function

1. The supplier requests a new delivery order through Supplier Client.
2. The Supplier Client forwards the request to the Supplier Business Object.
3. The Supplier Business Object sends the order request to the Delivery Order Business Object
4. The Delivery Order Business Object selects a delivery order by querying the Orders Server.
5. The Delivery Order Business Object sends the delivery order back to the Supplier Business Object
6. The Supplier Business Object checks whether the items are available in Inventory Server.

7. The Supplier Business Object sends the order status to the Supplier Client to display the delivery order and inventory information to supplier.

4.1.4. Confirm Shipment

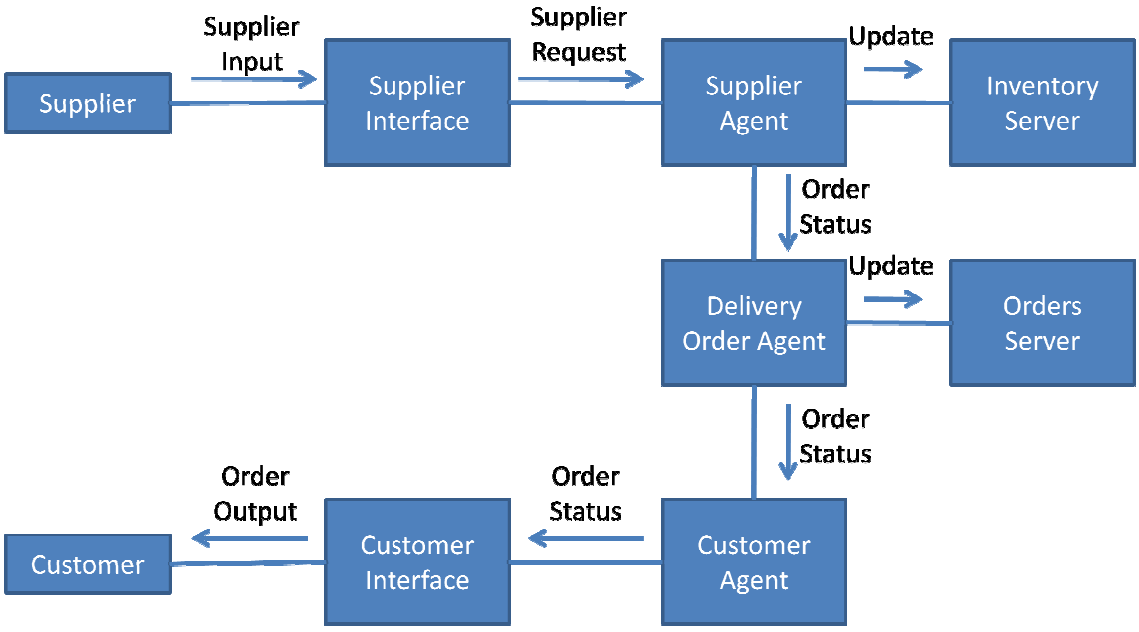


Figure 38: Confirm Shipment Function

1. The supplier inputs the shipping information.
2. The Supplier Client sends the supplier request to the Supplier Business Object.
3. The Supplier Business Object updates the inventory stored at the Inventory Server.
4. The Supplier Business Object sends the order status to the Delivery Order Business Object.
5. The Delivery Order Business Object updates order status the Orders Server.
6. The Delivery Order Business Object sends the order status to the Customer Business Object.
7. The Customer Business Object forwards the order status to the Customer Client to display to the customer.

4.1.5. Confirm Delivery

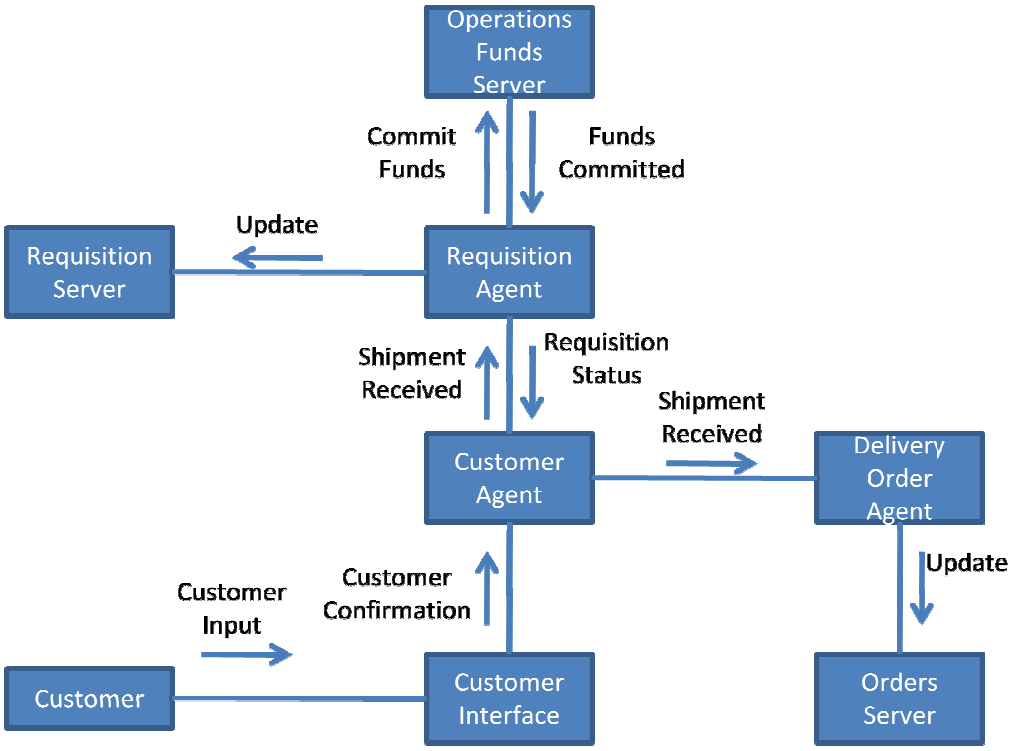


Figure 39: Confirm Delivery Function

1. Customer uses Customer Client to send delivery confirmation to the Customer Business Object.
2. The Customer Business Object sends a Shipment Received message to the Delivery Order Business Object.
3. The Delivery Order Business Object updates the status at the Orders Server.
4. The Customer Business Object sends a Shipment Received message to the Requisition Business Object.
5. The Requisition Business Object updates the status of the requisition stored at the Requisition Server.
6. The Requisition Business Object commits the funds for this requisition with the Operations Funds Server.

4.1.6. Send Invoice:

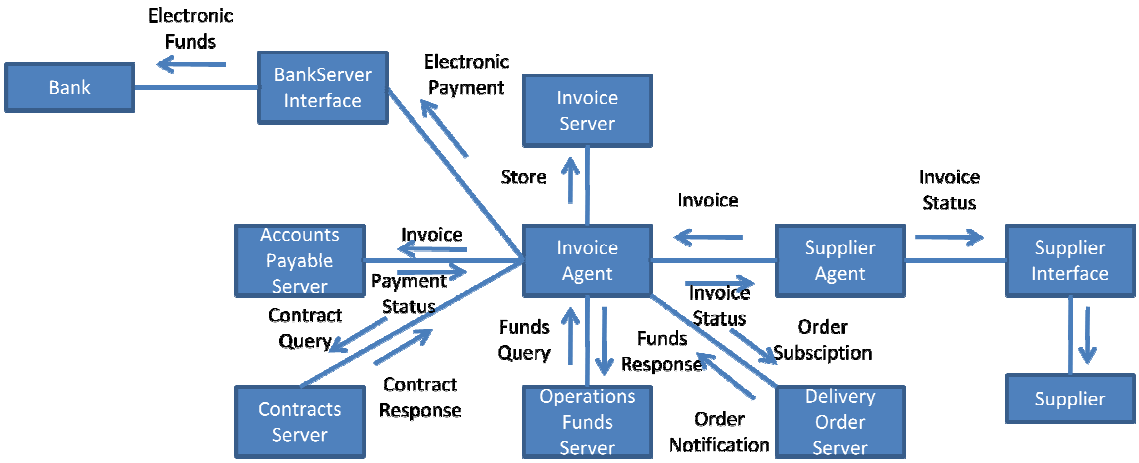


Figure 40: Send Invoice Function

1. The Supplier Business Object sends the invoice information to the Invoice Business Object.
2. The Invoice Business Object subscribes to the Delivery Order Business Object.
3. The Delivery Order Business Object notifies the Invoice Business Object that the goods have been received.
4. The Invoice Business Object sends a contract query to the Contracts Server.
5. The Contracts Server confirms the contract.
6. The invoice Business Object sends a funds query to the Operation Funds Server.
7. The Operation Funds Server confirms that the funds are available and committed.
8. The Invoice Business Object sends the invoice to the Accounts Payable Server.
9. The Accounts Payable Server sends the payment status to the Invoice Business Object.
10. The Invoice Business Object stores the invoice at the Invoice Server.
11. The Invoice Business Object sends the electronic payment to the customer's bank via the Bank Server Client.
12. Bank Server Client sends the electronic funds to the customer's bank for payment to the supplier.
13. The Invoice Business Object sends the invoice status to the Supplier Business Object.
14. The Supplier Business Object sends the invoice status to the Supplier Client.
15. The Supplier Client displays the invoice status to the supplier.

4.2. Structure

In the previous section, we examined all function of the E-Commerce System. In this part, we will model E-Commerce System by using Architecture Analysis and Design Language. With AADL model of E-Commerce System, we will easily control the data port of each component in the system.

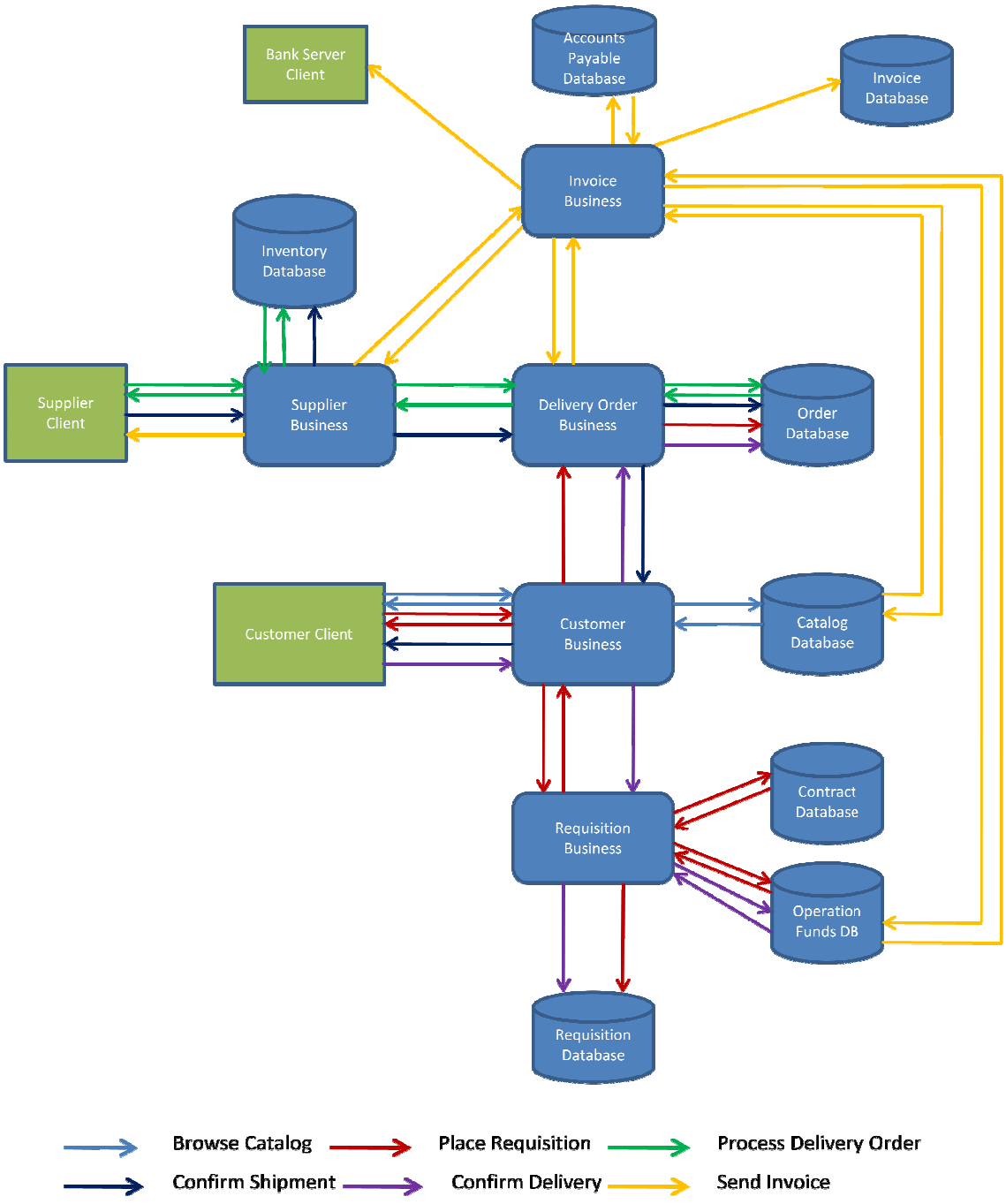


Figure 41: All function of E-Commerce System

4.2.1. Browse Catalog:

In the first function, Browse Catalog, the overview of this function can be described using AADL graphical type as following:

Figure 42: Browse Catalog in AADL

4.2.1.1. System Definition

The first component we describe in the system is Customer Client component. This component have two data ports: the first data port oCatalogBrowse is output data port to send a request string to Customer Business Object and the second one iCatalogResponse is input data port to receive the response data back from the Customer Business Object. In the E-Commerce System, almost all data ports are event data port because they need specific event to trigger sending or receiving data. Component Customer Client can be described by AADL text type as following:

```
system CustomerClient
      features
            oCatalogBrowse: out event data port string;
            iCatalogResponse: in event data port dCatalog.reg;
end CustomerClient;
```

The next component is the Customer Business Object component. Customer Business Object has 4 data ports: two input ports and two output ports. First, the Customer Business Object receives the catalog browsing request through the input event data port iCatalogBrowse. After that, it will make a query to Catalog Database through the output event data port oCatalogQuery and get the response by input event data port iCatalogResult. Finally, the data will be sent back to Customer Client using output event data port oCatalogResponse.

```
system CustomerBusiness
      features
            iCatalogBrowse: in event data port string;
```

```
            oCatalogQuery: out event data port string;
            iCatalogResult: in event data port dCatalog.reg;
            oCatalogResponse: out event data port dCatalog.reg;
end CustomerBusiness;
```

The last component in this Browse Catalog function is Catalog Database (CatalogDB). CatalogDB receives the browse catalog query from Customer Business Object and returns the catalog data to the Customer Client.

```
system CatalogDB
      features
            iCatalogQuery: in event data port string;
            oCatalogResult: out event data port dCatalog.reg;
end CatalogDB;
```

4.2.1.2. Data Definition

The data use in this function is string for making the query and the result is dCatalog with the structure is dCatalog.reg (regular implementation).

```
data dCatalog
end dCatalog;
data implementation dCatalog.reg
      subcomponents
            itemID: int;
            itemDescription: string;
            unitCost: real;
end dCatalog.reg;
```

4.2.2. Place Requisition

The graphic model of this function is as following:

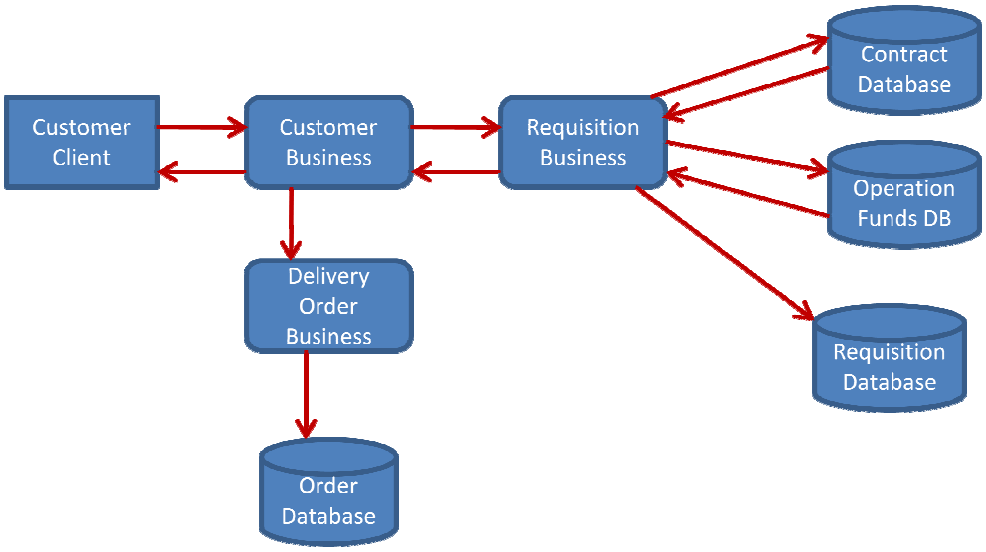


Figure 43: Place Requisition in AADL

4.2.2.1. System Definition

In the Place Requisition function, Customer Client has two more ports: output data port oRequisitionPlace and input data port iRequisitionPlaceResponse. Customer will use client to send his selected item through oRequisitionPlace output port and then will get the status of his requisition through iRequisitionPlaceResponse input port.

```
system CustomerClient
      features
            oRequisitionPlace: out event data port dSelectedItem.reg;
            iRequisitionPlaceResponse: in event data port dRequisition.reg;
end CustomerClient;
```

CustomerBusiness component also has more ports: 2 input ports and 3 output ports.

```
system CustomerBusiness
      features
            iRequisitionPlace: in event data port dSelectedItem.reg;
            oRequisitionPlacePass: out event data port dSelectedItem.reg;
            iRequisitionPlaceResult: in event data port dRequisition.reg;
            oPurchaseRequest: out event data port dDeliveryOrder.reg;
```

```
            oRequisitionPlaceResponse: out event data port
dRequisition.reg;
end CustomerBusiness;
```

This is the data port of RequisitionBusiness component.

```
system RequisitionBusiness
      features
            iRequisitionPlacePass: in event data port dSelectedItem.reg;
            oContractQuery: out event data port dCustom.PR2;
            iContractResult: in event data port dContract.reg;
            oFundReserve: out event data port dCustom.PR3;
            iFundReserveResult: in event data port dOperationFunds.reg;
            oRequisitionStore: out event data port dRequisition.reg;
            oRequisitionPlaceResult: out event data port dRequisition.reg;
end RequisitionBusiness;
```

Next is the data components.

ContractDB component:

```
system ContractDB
      features
            iContractQuery: in event data port dCustom.PR2;
            oContractResult: out event data port dContract.reg;
end ContractDB;
```

OperationFundsDB component:

```
system OperationFundsDB
      features
            iFundReserve: in event data port dCustom.PR3;
            oFundReserveResult: out event data port dOperationFunds.reg;
end OperationFundsDB;
```

RequisitionDB component:

```
system RequisitionDB
      features
```

```
            iRequisitionStore: in event data port dRequisition.reg;
end RequisitionDB;
```

OrderDB component:

```
system OrderDB
      features
            iDeliveryOrderCreate: in event data port dDeliveryOrder.reg;
end OrderDB;
```

Finally, this is the structure of DeliveryOrderBusiness component:

```
system DeliveryOrderBusiness
      features
            iPurchaseRequest: in event data port dDeliveryOrder.reg;
            oDeliveryOrderCreate: out event data port dDeliveryOrder.reg;
end DeliveryOrderBusiness;
```

4.2.2.2. Data Definition

In the Place Requisition function, we need to use some custom structure of data so we call it dCustom.

```
data dCustom
end dCustom;
```

With this custom structure data, we define two internal structures (two implementations) of dCustom. First implementation is dCustom.PR2.

```
data implementation dCustom.PR2
      subcomponents
            customerID: int;
            supplierID: int;
end dCustom.PR2;
```

Second implementation is dCustom.PR3.

```
data implementation dCustom.PR3
      subcomponents
```

```
            contractID: int;
            reservedFunds: real;
end dCustom.PR3;
```

The customer selects items they want to buy and send it to Customer Business to request the creation of requisition. Structure of selected item data is as following:

```
data dSelectedItem
end dSelectedItem;
data implementation dSelectedItem.reg
      subcomponents
            itemID: int;
            quantity: real;
            unitCost: real;
end dSelectedItem.reg;
```

After the Requisition Business has sent a contract query to the Contract Server, the Contract Server returns the contracts data with following structure.

```
data dContract
end dContract;

data implementation dContract.reg
      subcomponents
            contractID: int;
            maxPurchase: real;
end dContract.reg;
```

Operations Funds data structure:

```
data dOperationFunds
end dOperationFunds;

data implementation dOperationFunds.reg
      subcomponents
            operationFundsID: int;
            totalFunds: real;
            committedFunds: real
```

```
            reservedFunds: real;
end dOperationFunds.reg;
```

Requisition data structure:

```
data dRequisition
end dRequisition;

data implementation dRequisition.reg
      subcomponents
            requisitionID: int;
            amount: real;
            status: string;
end dRequisition.reg;
```

4.2.3. Process Delivery Order

The graphic model of this function is as following:

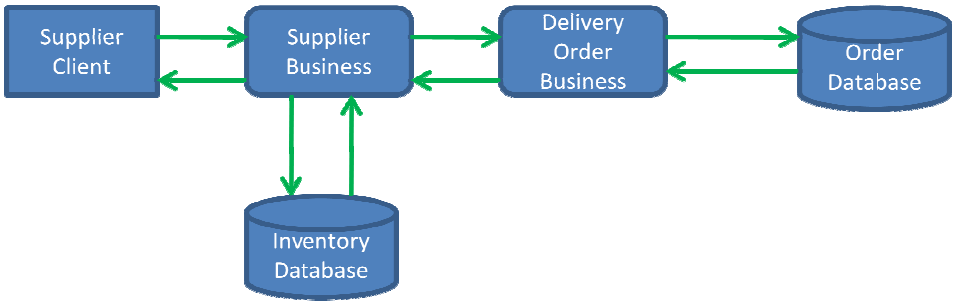


Figure 44: Process Delivery Order in AADL

4.2.3.1. System Definition

Supplier Client

```
system SupplierClient
      features
            oRequestNewDeliveryOrder: out event data port string;
            iRequestNewDeliveryOrderResponse: in event data port
dInventory.reg;
```

```
end SupplierClient;
```

Supplier Business

```
system SupplierBusiness
      features
            iRequestNewDeliveryOrder: in event data port string;
            oForwardRequest: out event data port string;
            iNewDeliveryResponse: in event data port dDeliveryOrder.reg[];
            oCheckInventory: out event data port dDeliveryOrder.reg ;
            iCheckInventoryResult: in event data port dInventory.reg[];
            oRequestNewDeliveryOrderResponse: out event data port
dInventory.reg;
end SupplierBusiness;
```

Delivery Order Business

```
system DeliveryOrderBusiness
      features
            iForwardRequest: in event data port string;
            oQueryNewDeliveryOrder: out event data port string;
            iNewDeliveryOrderResult: in event data port
dDeliveryOrder.reg[];
            oNewDeliveryResponse: out event data port dDeliveryOrder.reg[];
end DeliveryOrderBusiness;
```

Orders Database

```
system OrdersDB
      features
            iQueryNewDeliveryOrder: in event data port string;
            oNewDeliveryOrderResult: out event data port
dDeliveryOrder.reg[];
end OrdersDB;
```

Inventory Database

```
system InventoryDB
      features
```

```
            iCheckInventory: in event data port dDeliveryOrder.reg;
            oCheckInventoryResult: out event data port dInventory.reg[];
end InventoryDB;
```

4.2.3.2. Data Definition

Delivery Order data

```
data dDeliveryOrder
end dDeliveryOrder;

data implementation dDeliveryOrder.reg
      subcomponents
            orderId: int;
            plannedShipDate: date;
            actualShipDate: date;
            creationDate: date;
            orderStatus: string;
            amountDue: real;
            receivedDate: date;
end dDeliveryOrder.reg;
```

Inventory data

```
data dInventory
end dInventory;
data implementation dInventory.reg
      subcomponents
            itemID: int;
            itemDescription: string;
            quantity: int;
            price: real;
            reorderTime: date;
end dInventory.reg;
```

4.2.4. Confirm Shipment

The graphic model of this function is as following:

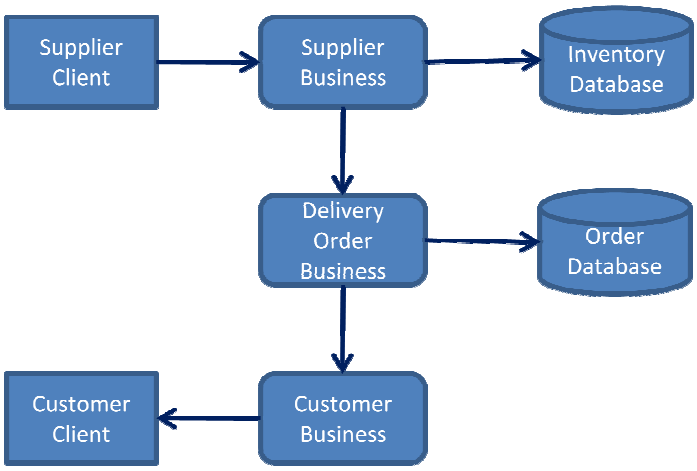


Figure 45: Confirm Shipment in AADL

4.2.4.1. System Definition

Supplier Client

```
system SupplierClient
      features
            oConfirmShipment: out event data port dCustom.CS1;
end SupplierClient;
```

Supplier Business

```
system SupplierBusiness
      features
            iConfirmShipment: in event data port dCustom.CS1;
            oUpdateInventory: out event data port dCustom.CS1;
            oSendOrderStatus: out event data port dCustom.CS1;
end SupplierBusiness;
```

Delivery Order Business

```
system DeliveryOrderBusiness
      features
            iSendOrderStatus: in event data port dCustom.CS1;
            oUpdateOrder: out event data port dCustom.CS1;
            oSendOrderStatusCustomer: out event data port dCustom.CS1;
```

```
end DeliveryOrderBusiness;
```

Order Database

```
system OrderDB
      features
            iUpdateOrder: in event data port dCustom.CS1;
end OrderDB;
```

Customer Business

```
system CustomerBusiness
      features
            iSendOrderStatusCustomer: in event data port dCustom.CS1;
            oDisplayOrderStatus: out event data port dCustom.CS1;
end CustomerBusiness;
```

Customer Client

```
system CustomerClient
      features
            iDisplayOrderStatus: in event data port dCustom.CS1;
end CustomerClient
```

Inventory Database

```
system InventoryDB
      features
            iUpdateInventory: in event data port dCustom.CS1;
end InventoryDB;
```

4.2.4.2. Data Definition

Another custom data: dCustom.CS1

```
data implementation dCustom.CS1
      subcomponents
            orderID: int;
            actualShipDate: date;
```

```
            orderStatus: string;
            itemID[]: int[];
            quantity[]: int[];
end dCustom.CS1;
```

4.2.5. Confirm Delivery

The overview of this function can be described using AADL graphical type as following.

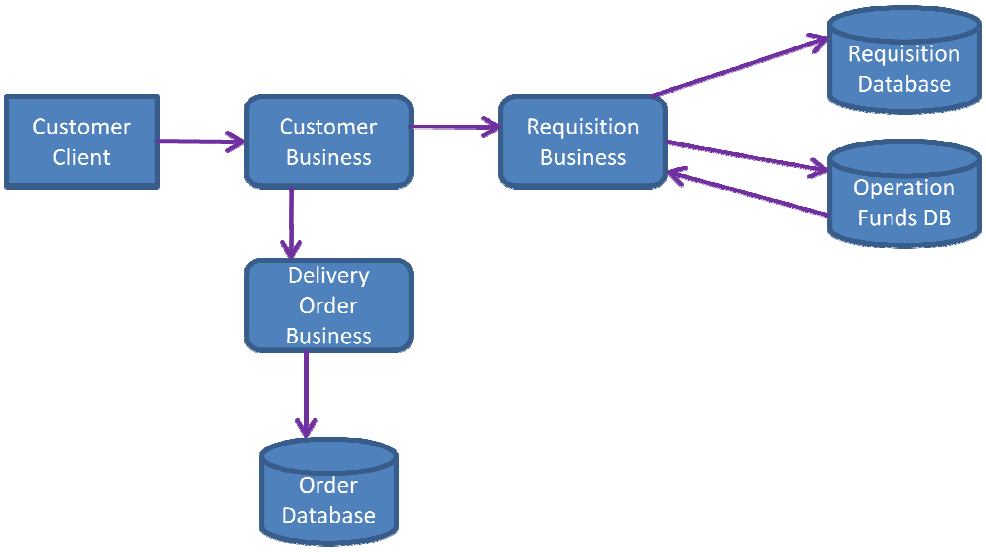


Figure 46: Confirm Delivery in AADL

4.2.5.1. System Definition

Customer Client

```
system CustomerClient
      features
            oConfirmDelivery: out event data port dCustom.CS1;
end CustomerClient;
```

Customer Business

```
system CustomerBusiness
      features
            iConfirmDelivery: in event data port dCustom.CS1;
```

```
            oForwardUpdateOrderStatus: out event data port dCustom.CS1;
            oForwardUpdateRequisition: out event data port
dRequisition.reg;
end CustomerBusiness;
```

Delivery Order Business

```
system DeliveryOrderBusiness
      features
            iForwardUpdateOrderStatus: in event data port dCustom.CS1;
            oUpdateOrderStatus: out event data port dCustom.CS1;
end DeliveryOrderBusiness;
```

Order Database

```
system OrderDB
      features
            iUpdateOrderStatus: in event data port dCustom.CS1;
end OrderDB;
```

Requisition Business

```
system RequisitionBusiness
      features
            iForwardUpdateRequisition: in event data port dRequisition.reg;
            oUpdateRequisition: out event data port dRequisition.reg;
            oCommitFund: out event data port dOperationFunds.reg;
            iCommitFundResult: in event data port dOperationFunds.reg;
end RequisitionBusiness;
```

Operation Funds Database

```
system OperationFundsDB
      features
            iCommitFund: in event data port dOperationFunds.reg;
            oCommitFundResult: out event data port dOperationFunds.reg;
end OperationFundsDB;
```

4.2.5.2. Data Definition

There is no new data using in this function.

4.2.6. Send Invoice

The overview of final function can be described using AADL graphical type as following.

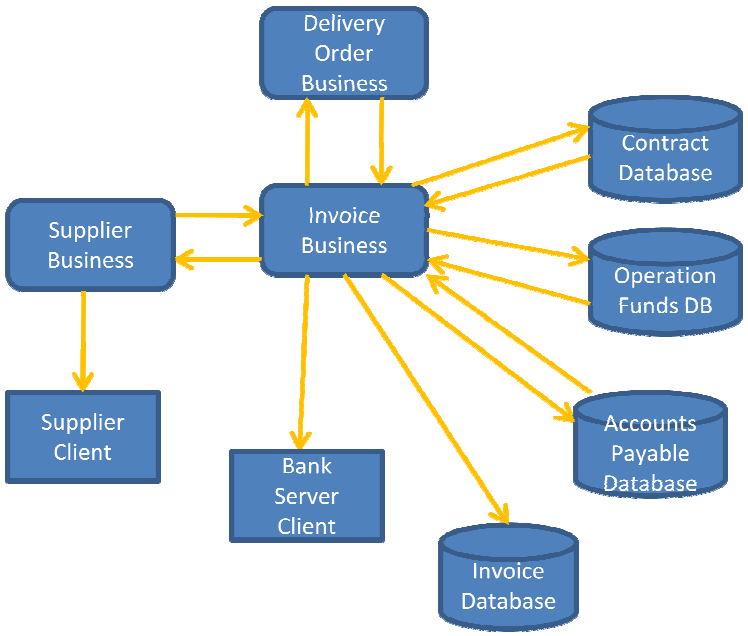


Figure 47: Send Invoice in AADL

4.2.6.1. System Definition

Supplier Business

```
system SupplierBusiness
      features
            oSendInvoice: out event data port dInvoice.reg;
            iSendInvoiceStatus: in event data port dInvoice.reg;
            oDisplayInvoiceStatus: out event data port dInvoice.reg;
end SupplierBusiness;
```

Supplier Client

```
system SupplierClient
      features
            iDisplayInvoiceStatus: in event data port dInvoice.reg;
end SupplierClient;
```

A new component that appears in this Send Invoice function is Invoice Business.

```
system InvoiceBusiness
      features
            iSendInvoice: in event data port dInvoice.reg;
            oCheckReceive: out event data port dCustom.CS1;
            iCheckReceiveResponse: in event data port dCustom.CS1;
            oCheckContract: out event data port dCustom.PR3;
            iCheckContractResponse: in event data port dContract.reg;
            oCheckFunds: out event data port dOperationFunds.reg;
            iCheckFundsResponse: in event data port dOperationFunds.reg;
            oAuthorizePayment: out event data port dInvoice.reg;
            iAuthorizePaymentResponse: in event data port dPayment.reg;
            oStoreInvoice: out event data port dInvoice.reg;
            oSendPaymentToCustomer: out event data port dPayment.reg;
            oSendInvoiceStatus: out event data port dInvoice.reg;
end
```

Delivery Order Business

```
system DeliveryOrderBusiness
      features
            iCheckReceive: in event data port dCustom.CS1;
            oCheckReceiveResponse: out event data port dCustom.CS1;
end DeliveryOrderBusiness;
```

Contract Database

```
system ContractDB
      features
            iCheckContract: in event data port dCustom.PR3;
            oCheckContractResponse: out event data port dContract.reg;
```

```
end ContractDB;
```

Operation Funds Database

```
system OperationFundsDB
      features
            iCheckFunds: in event data port dOperationFunds.reg;
            oCheckFundsResponse: out event data port dOperationFunds.reg;
end OperationFundsDB;
```

A new data component is Accounts Payable Database.

```
system AccountsPayableDB
      features
            iAuthorizePayment: in event data port dInvoice.reg;
            oAuthorizePaymentResponse: out event data port dPayment.reg;
end AccountsPayableDB;
```

Invoice Database

```
system InvoiceDB
      features
            iStoreInvoice: in event data port dInvoice.reg;
end InvoiceDB;
```

Bank Server Client

```
system BankServerClient
      features
            iSendPaymentToCustomer: in event data port dPayment.reg;
end BankServerClient;
```

4.2.6.2. Data Definition

Invoice data

```
data dInvoice
end dInvoice;
data implementation dInvoice.reg
```

```
      subcomponents
            invoiceID: int;
            amountDue: real;
            invoiceDate: date;
end dInvoice.reg;
```

Payment data

```
data dPayment
end dPayment;
data implementation dPayment.reg
      subcomponents
            paymentID: string;
            amount: real;
            date: date;
            status: string;
end dPayment.reg;
```

4.3. Updating type using E-Commerce System:

Although the Electronic Commerce System is satisfied the basic business of a commerce system, as mention before, E-Commerce System still need to updating. Some reasons can be fixing bugs, add new functions, or improve performance.

4.3.1. Updating Type 0 in E-Commerce System

Because of the characteristic of updating type 0, this updating type is not change external data flow of the updated component so we can ignore it during making non-stop upgrading of the web application.

4.3.2. Updating Type 1 in E-Commerce System

In the example for applying updating type 1 in E-Commerce System, we use the Browse Catalog use case. In the original of this function, customers will use client to send the request to Customer Business Object. The business object will retrieve catalog data in database and send back to the client.

Figure 48: Browse Catalog before updating

For this example, the update is that the Customer Business not only retrieves data from Catalog database but also check the Inventory database to response to customer not all catalog but only catalogs in stock.

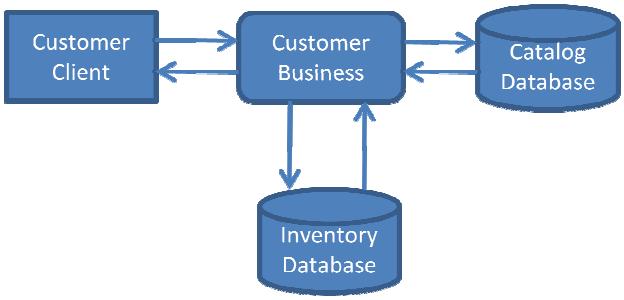


Figure 49: Browse Catalog after updating

The change to AADL description of Customer Business is two more ports oCheckInventory and iCheckInventoryResult:

```
system CustomerBusiness
      features
            iCatalogBrowse: in event data port string;
            oCatalogQuery: out event data port string;
            iCatalogResult: in event data port dCatalog.reg[];
            oCatalogResponse: out event data port dCatalog.reg[];
            oCheckInventory: out event data port dCatalog.reg;
            iCheckInventoryResult: in event data port bool;
end CustomerBusiness;
```

And the change to Inventory Database is two more port too:

```
system InventoryDB
      features
            iCheckInventory: in event data port dCatalog.reg;
            oCheckInventoryResult: out event data port bool;
end InventoryDB;
```

The Restricted Sequence Diagram of the Browse Catalog function is below:

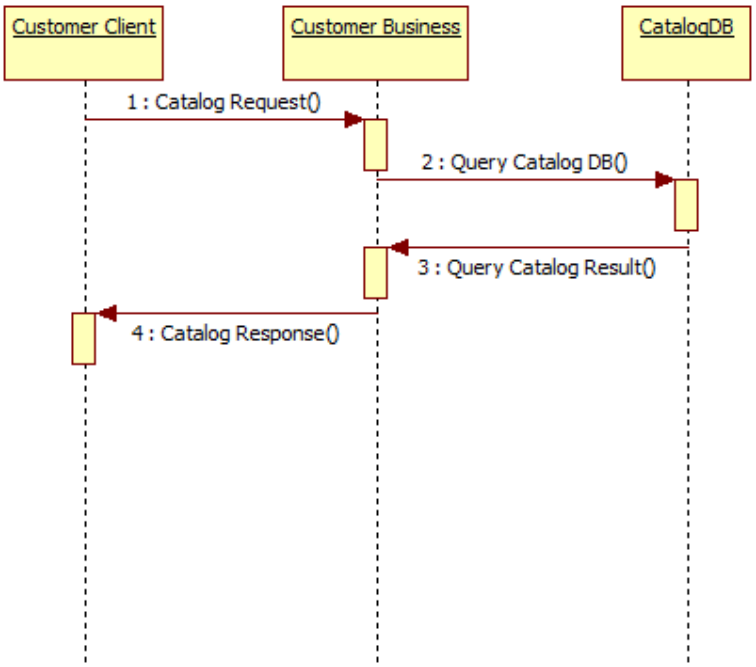


Figure 50: Browse Catalog Restricted Sequence Diagram before updating

Set of objects:

$O = \{CustomerClient, CustomerBusiness, CatalogDB\}$

Set of flows:

$$F = \{ f_1 < CustomerClient, CustomerBusiness >,$$
$$f_2 < CustomerBusiness, CatalogDB >,$$
$$f_3 < CatalogDB, CustomerBusiness >,$$
$$f_4 < CustomerBusiness, CustomerClient > \}$$

Orders:

$$\{< 1, f_1 >< 2, f_2 >< 3, f_3 >< 4, f_4 >\}$$

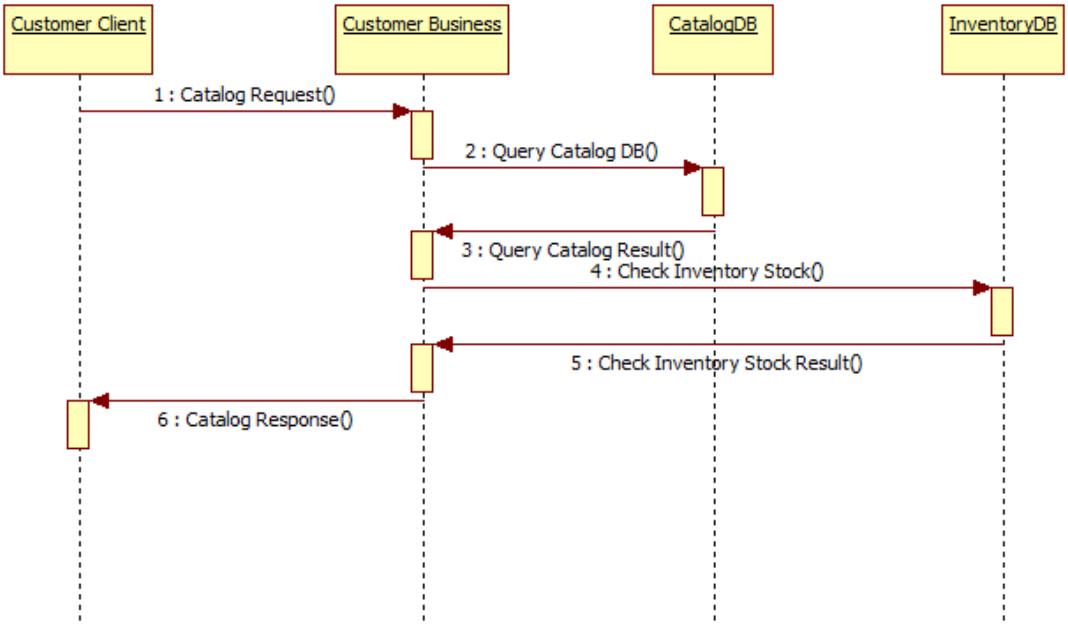


Figure 51: Browse Catalog Restricted Sequence Diagram after updating

Set of objects:

$$O = \{ CustomerClient, \quad CustomerBusiness, \quad CatalogDB, \quad \boldsymbol{InventoryDB} \}$$

Set of flows:

$$F = \{ f_1 < CustomerClient, CustomerBusiness >,$$
$$f_2 < CustomerBusiness, CatalogDB >,$$
$$f_3 < CatalogDB, CustomerBusiness >,$$
$$f_4 < CustomerBusiness, CustomerClient >,$$
$$\boldsymbol{f_5} < CustomerBusiness, InventoryDB >,$$
$$\boldsymbol{f_6} < InventoryDB, CustomerBusi\boldsymbol{ness} > \}$$

Orders:

$$\{< 1, f_1 >< 2, f_2 >< 3, f_3 >< 4, \boldsymbol{f_5} >< 5, \boldsymbol{f_6} >< 6, f_4 >\}$$

4.3.3. Updating Type 3 in E-Commerce System:

Updating type 3 is a specific case of updating type 2 so that we consider type 3 of updating. Part of Place Requisition AADL graphic is as following.

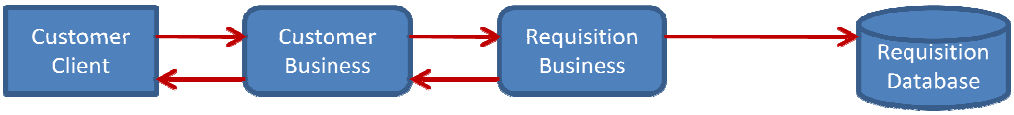


Figure 52: Place Requisition before update

In the original of Place Requisition function, Requisition Business only sends query to store new requisition to Requisition Database. In the example of using updating type 3, we want to confirm that storing requisition is successful so that we need a confirmation from Requisition Database.

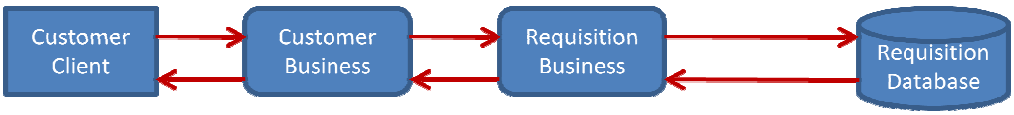


Figure 53: Place Requisition after update

```
system RequisitionBusiness
      features
            iRequisitionPlacePass: in event data port dSelectedItem.reg;
            oRequisitionStore: out event data port dRequisition.reg;
            iRequisitionStoreResult: in event data port bool;
            oRequisitionPlaceResult: out event data port dRequisition.reg;
end RequisitionBusiness;
```

```
system RequisitionDB
      features
            iRequisitionStore: in event data port dRequisition.reg;
            oRequisitionStoreResult: out event data port bool;
end RequisitionDB;
```

Restricted Sequence Diagram before update is described below.

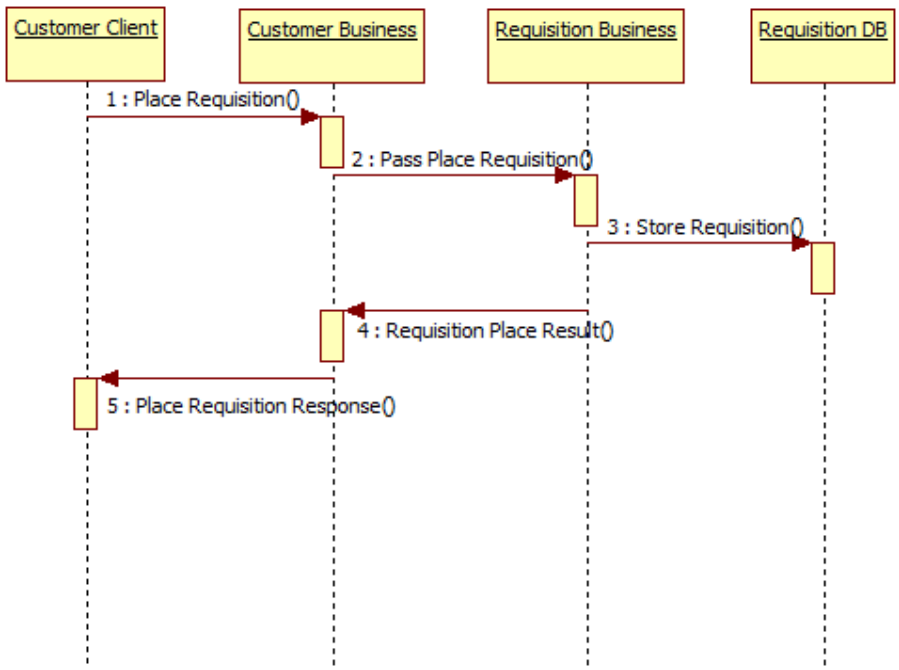


Figure 54: Place Requisition Restricted Sequence Diagram before update

Set of objects

$O = \{CustomerClient, CustomerBusiness, RequisitionBusiness, RequisitionDB\}$

Set of flows

$F = \{ f1 < CustomerClient, CustomerBusiness >,$
$f2 < CustomerBusiness, RequisitionBusiness >,$
$f3 < RequisitionBusiness, RequisitionDB >,$
$f4 < RequisitionBusiness, CustomerBusiness >,$
$f5 < CustomerBusiness, CustomerClient > \}$

Orders:

$\{< 1, f1 >< 2, f2 >< 3, f3 >< 4, f4 >< 5, f5 >\}$

And the diagram changed as following after updating.

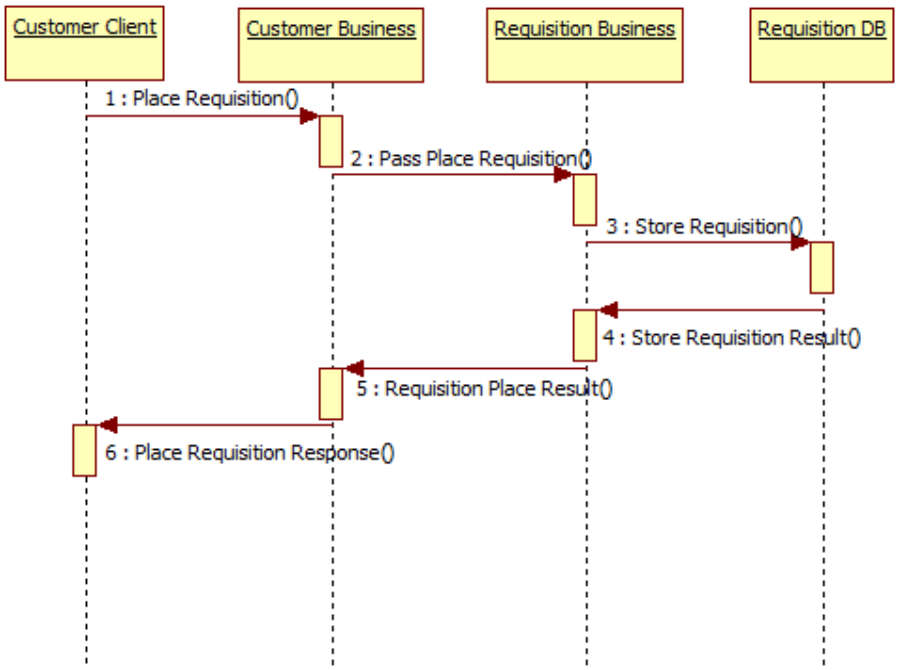


Figure 55: Place Requisition Restricted Sequence Diagram after update

Set of objects

$O = \{CustomerClient, CustomerBusiness, RequisitionBusiness, RequisitionDB\}$

Set of flows

$F: \{ f1 < CustomerClient\ , CustomerBusiness >,$

$f2 < CustomerBusiness, RequisitionBusiness >,$

$f3 < RequisitionBusiness, RequisitionDB >,$

$f4 < RequisitionBusiness, CustomerBusiness >,$

$f5 < CustomerBusiness, CustomerClient >,$

$\boldsymbol{f6} < RequisitionDB, RequisitionBusiness > \}$

Orders:

$\{< 1, f1 > < 2, f2 > < 3, f3 > < 4, f\boldsymbol{6} > < 5, f4 > < 6, f5 >\}$

Identify: <3,f3> and the new <4,f6>, the source and destination of f3 maybe same or in reverse order to f6

Chapter 5:

Evaluation

5.1. Upgrading web application

We all know that the web application is very large and has many components. These components can be worked independent to other components. However, in some specific case, these component results can also affect to other components activities. So the effective way of making non-stop upgrading is ensuring that the data after making an update is not effect to other update. This will guarantee that the upgrading process will be done successfully and safely.

There are 3 issues we meet when making a non-stop upgrading of system:

- Typical change might be achieved by some upgrading operations.
- During each upgrade operation, service progresses must not corrupted.
- No data missing.

In the reality, each time we need to make the system maintenance, we often need to perform a sequence of updating process. From this viewpoint, almost all maintenance process can be considered as the upgrading process.

Assume we apply a sequence of updating U_1, U_2, U_3... to the system S.

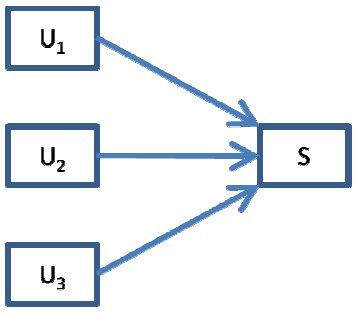


Figure 56: Upgrading system S

Because this is a web application system, we can choose the order of updating operation. There will have many ways to finish the upgrading of system S.

For example we have n operations of upgrade then number of updating path becomes n! unless they collapse.

5.2. Step to make a non-stop upgrading

Non-stop upgrading means that the service may postpone but never terminated during upgrading. In order to guarantee that the upgrading process does not change or ignored the overall system activities, we should follow these steps when doing the upgrading:

First of all, we will use the AADL description to describe the data ports of the updating component. In this step, we will have the general view about the data ports using by updating components. We also identify the data sending out and in of these components.

Next, we need to describe the data flow between these components by using Restricted Sequence Diagram.

After that, we will use AADL description and Restricted Sequence Diagram to model the system after updating. Depend on these model, we can determine what type of the update.

With each of specific update, we can define what data the update changes and we can determine the order of upgrading process or know that the upgrading process cannot be done.

5.3. Upgrading achievement

Non-stop upgrading can be achieved by satisfy two viewpoints in our architecture:

- Static viewpoint: data is not missing or corruption after upgrading.
- Dynamic viewpoint: the overall message flows are correct.

5.4. Concrete example

Figure 33 describes a system with 3 components O_1, O_2, O_3, and there are the data flows f_1, f_2 between them. This is the original state of the system S, we called this state S_0.

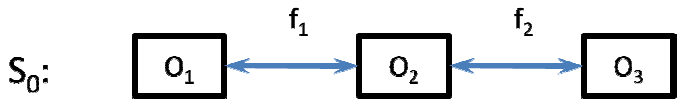


Figure 57: Upgrading System State 0

Assume that we want to upgrade S_0 in order to extend two functions. One is implemented in 0_4 and another in 0_5. First update is type 1, making a new connection from O_1 to O_4. Second update is type 4, inserting a new object O_5 between O_2 and O_3.

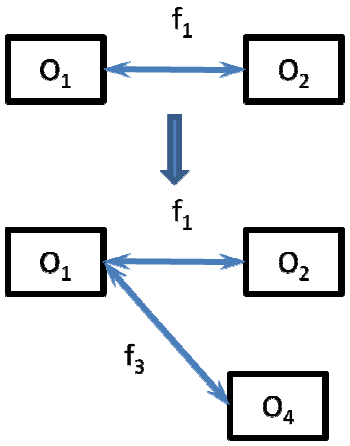


Figure 58: First part of upgrading S

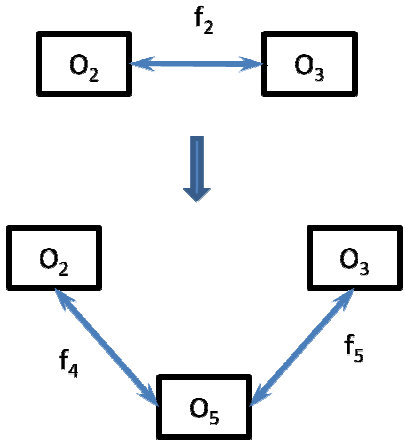


Figure 59: Second part of upgrading S

There will have two middle states of system S before finishing upgrading. We assume that S_1 is the state after finishing type 1 update and S_2 is the state after making type 4 update.

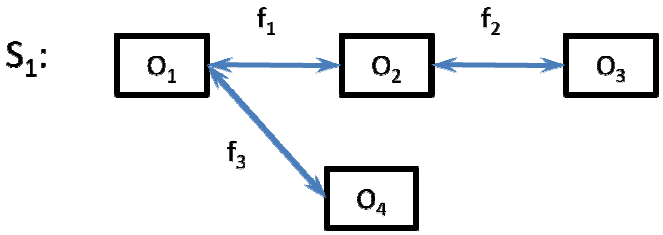


Figure 60: Upgrading System State 1

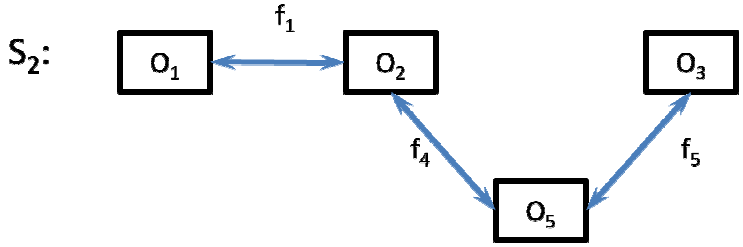


Figure 61: Upgrading System State 2

We must complete two previous states to finish this upgrading requirement. The final model of upgrading the system S is S_{12} and is described in Figure 38.

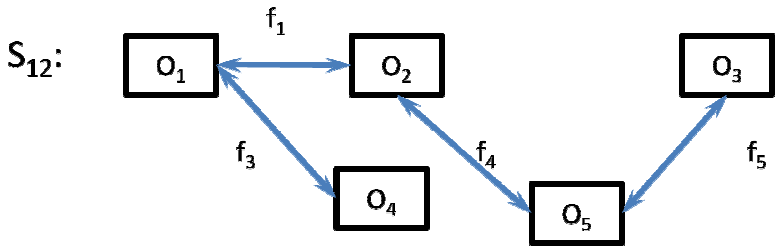


Figure 62: Finish state of upgrading system S

So we have two paths to reach the complete upgrading state of system S.

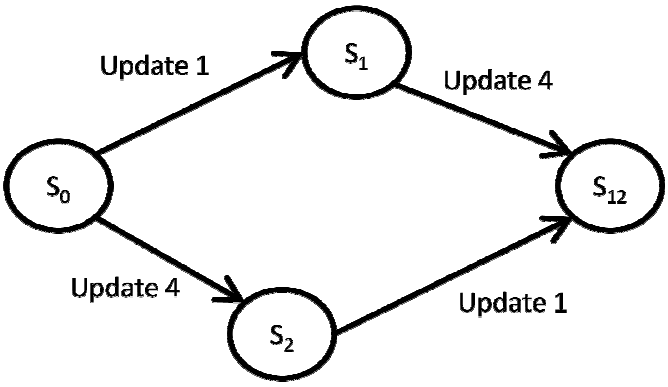


Figure 63: Upgrade path

The update is possible if in each state of updating process; the data can be sent without confliction. For example, after making type 1 update, the system is S1 in Figure 36. This update is possible if the data can be sent without change from $O_1 \rightarrow O_2 \rightarrow O_3$ and vice versa.

Because of time restriction, we cannot clarify detail cases of data confliction and all the combination of updating type. This works should be done in the future.

Chapter 6:

Conclusion and future works

The task of making non-stop upgrading is an important and challenging task in software development. This problem relates to both local and web-based software development. Because of characteristic of each updating type, this task is divided into two independent tasks.

In local software development domain, the entire program is running in local computer so that the upgrading task is too different to the upgrading task of web-based applications. In web application, the system is very large, it uses database to store the data and address of components or data is not gather in one specific address. In a web application, two databases can be placed at more than one address.

These different points make the mechanism for making non-stop upgrading web application and local application different. There are many researches to achieve the goal for both web application and local application. However, these researches for non-stop upgrading web application only focus only the technique for deployment in real application. In other words, these researches focus on programming technique to realize non-stop upgrading web application.

In this dissertation, we not focus on these techniques but we focus on finding a good method to describe the system and data using in the system. We addressed a solution based on Interceptor architecture and component-based approach. With our method, we have a simple and clear view of the data using in the system. Furthermore we use a new way to control the upgrading process of the system so that we can achieve the goal of the research: non-stop upgrading of web application.

First we need to describe the system formally to control the data in the system in order to manage upgrading work. So we define a new way to model the system using the combination of AADL and RSD. With our method, we can easily understand the behavior of the system, including data flow and order of the data sending in the system.

Secondly we define an upgrading as a sequence of updating operations and categorize them into five patterns. Each type of update can change the system differently and can affect the time we upgrade the system. The first case is internal changes of components. The second is the case that a new component is added into the system. Next case is that we add new connection from two existed components. Final case is that we have a new component between two existed components.

Next we showed a mechanism for using Interceptor pattern. The Interceptor architectural pattern is one of pattern-oriented software architecture introduced in as a pattern for concurrent and networked objects. The Interceptor architectural pattern allows services to be added transparently to a framework and triggered automatically when certain events occur. So we can apply this architecture to control data flow in the system to achieve the goal.

Our solution has many advantages to traditional approaches such that we can control the consistency of the data in the system as well as the overall system data flow and our solution has the flexibility and extendibility for development.

The most important one is consistency, which means the data in the system after upgrading is not conflict with other process of the system. Our mechanism also has flexibility and extendibility for improving in the future.

In many cases, the non-stop upgrading can be achieved without any problems. However, there will be some situations that the data is conflicted between updating processes or the data flow can be incorrect after upgrading. We are now engaging in examining all possible upgrading case. In the future, we need to consider all of the cases to clarify detail of data confliction may be existed. And the most important and difficult part is that apply the theory into practical, we need to work a lot in the future to apply our architecture into real application.

Reference

1. *A simple equation: IT on = Business on.* **Parker, S.** s.l. : The IT Journal, Hewlett Packard, 2001.

2. **Larman, Craig.** *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design.* s.l. : Prentice Hall PTR, 1998. ISBN 0-13-78880-7.

3. **Koskinen, Jussi.** *Software Maintenance Costs.* Information Technology Research Institute, Finland : s.n., 2003.

4. **Gomaa, Hassan.** *E-Commerce: Designing Concurrent, Distributed, and Real-time applications with UML.* s.l. : Addison-Wesley, July 2001. ISBN 0-201-65793-7, Second Printing.

5. **Bell, Donald.** UML basics: The sequence diagram. *IBM developerWorks web site.* [Online] IBM, 2004. http://www.ibm.com/developerworks/rational/library/3101.html.

6. **Ambler, Scott W.** Introduction to UML 2 Sequence Diagrams. *Agile Modeling (AM).* [Online] Ambysoft Inc, 2003-2010. http://www.agilemodeling.com/artifacts/sequenceDiagram.htm.

7. **Peter H. Feiler, David P. Gluch, and John J. Hudak.** *The Architecture Analysis & Design Language (AADL): An Introduction.* February 2006. CMU/SEI-2006-TN-011.

8. Sequence diagram. *Wikipedia, the free encyclopedia.* [Online] http://en.wikipedia.org/wiki/Sequence_diagram.

9. *ROC-1: Hardware Support for Recovery-Oriented Computing.* **David Oppenheimer, Aaron Brown, James Beck, Daniel Hettena, Jon Kuroda,Noah Treuhaft, David A. Patterson, Fellow, IEEE, and Katherine Yelick, Member, IEEE.** s.l. : IEEE TRANSACTIONS ON COMPUTERS, 2002, Vol. 51.

10. *Practical Dynamic Software Updating for C.* **Iulian Neamtiu, Michael Hicks, Gareth Stoyle, Manuel Oriol.** Ottawa, Ontario, Canada : ACM, 2006.

11. **Douglas Schmidt, Michael Stal, Hans Rohnert and Frank Buschmann.** *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, Volume 2.* s.l. : John Wiley & Sons, 2000. ISBN 0471606952.

12. *Online Non-stop Software Updating Using Replicated Execution Blocks.* **Kuo-Feng Ssu, Hewijin Christine Jiau.** Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan 701 : International Computer Software and Applications Conference, 2000.

13. *Mutatis Mutandis: Safe and Predictable Dynamic Software Updating.* **Gareth Stoyle, Michael Hicks, Gavin Bierman, Peter Sewell, Iulian Neamtiu.** California, USA : ACM, 2005.

14. *Implementation of Non-stop Software Update for Client-Server Applications.* **Wen-Kang Wei, Kuo-Feng Ssu, Hewijin Christine Jiau.** Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan 701 : Computer Software and Applications Conference, 2003.

15. *Extending Message-Oriented Middleware using Interception.* **Edward Curry, Desmond Chambers, and Gerard Lyons.** Department of Information Technology, National University of Ireland, Galway, Ireland : s.n.

16. **Christopher M. Hayden, Eric A. Hardisty, Michael Hicks, and Jeffrey S. Foster.** *Efficient systematic testing for dynamically updatable software.* s.l. : ACM, 2009. ISBN 978-1-60558-723-3 .

17. *Dynamic Software Updating.* **Michael Hicks and Scott Nettles.** s.l. : ACM Transactions on Programming Languages and Systems.

18. **Inderjeet Singh, Beth Stearns, Mark Johnson, and the Enterprise Team.** *Designing Enterprise Applications with the J2EE Platform, Second Edition.* s.l. : Addison-Wesley, 2002. ISBN 0-201-78790-3.

19. *CHARACTERISTICS OF WEB APPLICATIONS THAT AFFECT USABILITY: A REVIEW.* **Vince Bruno, Audrey Tam, James Thom.** Canberra, Australia : OZCHI 2005, 2005.

20. *Aspects in the industry standard AADL.* **Dionisio de Niz and Peter H. Feiler.** New York, USA : ACM, 2007. ISBN:978-1-59593-658-5.