Supplement on Buyer/Seller Purchase Order Example

Notes for Course material

“Model Based System Development”

INF5120 – Spring 2008
### Table of Contents

**Part V**

Notes for Course material ............................................................................................................ 1

“Model Based System Development” .......................................................................................... 1

INF5120 – Spring 2008 ................................................................................................................. 1

Executive Summary ...................................................................................................................... 4

### I COMET-S - Overview ........................................................................................................... 5

I.1 The different models in COMET-S ................................................................................. 5

### II COMET - Business model ............................................................................................. 6

II.1 BMM – Business Motivation Metamodel ....................................................................... 6

   II.1.1 Example – BMM – Buyer/Seller ...................................................................... 6

   II.1.2 Possible Mapping between BMM and PIM4Agent ........................................... 9

II.2 BPMN - The Business Process Modeling Notation ....................................................... 12

   II.2.1 BPMN example for Buyer_Seller ............................................................... 12

   II.2.2 Mapping between BPMN and UPMS .......................................................... 16

II.3 COMET Business model overview ............................................................................... 21

   II.3.1 COMET Business model overview .............................................................. 21

   II.3.2 The business resource model – UML Class model and OWL ...................... 22

II.4 Work analysis refinement model (WARM) ................................................................... 24

   II.4.1 Modeling objectives .................................................................................... 24

   II.4.2 Methods and techniques .......................................................................... 25

   II.4.3 WARM implementation ............................................................................ 26

### III COMET - Requirements model ..................................................................................... 29

III.1 Introduction – COMET Requirements model ............................................................... 29

III.2 COMET - Requirements model to service architecture model transformation .......... 29

   III.2.1 Requirements model to architecture model transformation ....................... 29

III.3 Requirements model example ....................................................................................... 32

   III.3.1 Product Vision and Description .................................................................. 32

   III.3.2 Introduction ............................................................................................. 32

   III.3.3 Service information model – Class Diagrams ............................................. 32

### IV COMET-S - Service architecture model (UPMS ) ......................................................... 35

IV.1 Introduction .................................................................................................................... 35

   IV.1.1 Service Profile and meta-model .................................................................. 35

IV.2 Buyer_Seller examples .................................................................................................. 36

IV.3 COMET - Service architecture metamodel ................................................................... 41

### V COMET-S - Platform specific model............................................................................... 42

V.1 Introduction .................................................................................................................... 42

V.2 Platform profile model – JEE Meta-model ...................................................................... 42

   V.2.1 Goals ........................................................................................................ 42

### VI JEE43

VI.1 JEE Meta-model ................................................................................................................ 43

   VI.1.1 Introduction ............................................................................................. 43

   VI.1.2 Distributed Multi-tiered Applications ...................................................... 43

   VI.1.3 Java EE Components .............................................................................. 44

   VI.1.4 Business Components ............................................................................ 44

   VI.1.5 Enterprise Information System Tier .......................................................... 44

   VI.2 JEE Notation and UML Profile .......................................................................... 45

   VI.2.1 Enterprise JavaBeans Technology ............................................................ 45
Executive Summary

This document presents “State-of-the art for Interoperability architecture approaches” with a focus on “Model driven and dynamic, federated enterprise interoperability architectures and interoperability for non-functional aspects” – as in the title of INTEROP WP9/JR4 – and later for the DAP domain.

This document is a supplement part V to a series of four part of lecture notes for the course INF5120, Model Based System Development, for the spring of 2008, as follows.

Part I – MDE – Model Driven Engineering
Part II – SOA - Service Oriented Architectures
Part III – MDE for SOA
Part IV – MDI – Model Driven Interoperability
Part V – Supplement on Buyer/Seller PurchaseOrder example – for part III

Part I focuses on - Model Driven Engineering – with an introduction to principles of metamodeling and related MDE ated standards and technologies, in particular related to MDA and Eclipse EMF. The relationship between UML profiles and Domain Specific Languages (DSL) is introduced, as well as an overview of various model transformation technologies including model-to-model with ATL and model-to-text with MOFScript. It is shown how method engineering can be supported by the OMG SPEM standard and the Eclipse EPF framework.

Part II focuses on SOA - Service Oriented Architectures – with a basis in concepts for service oriented computing, with a special emphasis on technologies for web services with XML, WSDL and BPEL. The basis technologies for the semantic web is also introduced with RDF and OWL, and semantic web services with OWL-S and WSMO. A last section presents agent-oriented computing with multi agent systems (MAS) and a platform independent model for agents (PIM4Agents).

Part III focuses on MDE for SOA - Model Driven Engineering for Service Oriented Architectures – and applies the principles of model driven engineering to service oriented architectures. The starting point for this approach is the COMET methodology used in previous INF5120 courses, this year enhanced to become COMET-S for Services through the use of new standard UML profiles and metamodels. The Business model uses in particular BMM (Business Motivation Metamodel, BPMN (Business Process Modelling Notation). The Requirements model supports mappings from use cases to services definitions. The service architecture model uses the new UPMS (UML Profile and Metamodel for Services). The platform specific model will vary depending on the target platform. The course has been using the JEE platform as reflected in the Oblig exercises in the course.

Part IV focuses on MDI - Model Driven Interoperability – and illustrates how a model driven approach can be applied to the problem domain of interoperability through the use of horizontal mappings and transformations. The approach to this is illustrated with the AIF (ATHENA Interoperability Framework) and the AIM (ATHENA Interoperability Methodology) and a set of articles on MDI.

Part V – Supplement on Buyer/Seller PurchaseOrder example provides a reference example based on the material in part III – applied to a Buyer/Seller PurchaseOrder example.
I COMET-S - Overview

I.1 The different models in COMET-S

This example follows the COMET-S steps and models from part III, as follows:
1) Business model – with BMM and BPMN and resource/ontology model
2) Requirements model
3) Service architecture model – with UPMS SOAPro
4) JEE platform specific model – with ATL and MOFScript
II COMET - Business model

II.1 BMM – Business Motivation Metamodel

The Business Motivation Model provides a scheme or structure for developing, communicating, and managing business plans in an organized manner. Specifically, the Business Motivation Model does all of the following:

- It identifies factors that motivate the establishing of business plans.
- It identifies and defines the elements of business plans.
- It indicates how all these factors and elements inter-relate.

Among these elements are ones that provide governance for and guidance to the business — Business Policies and Business Rules.

II.1.1 Example – BMM – Buyer/Seller

Examples of goal, objective, vision can be seen in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buyer</td>
<td>Goal</td>
<td>get the wanted product with sound price and good quality</td>
</tr>
<tr>
<td>Buyer</td>
<td>Objective</td>
<td>find the proper merchant</td>
</tr>
<tr>
<td>Buyer</td>
<td>Objective</td>
<td>define a budget for the product</td>
</tr>
<tr>
<td>Buyer</td>
<td>Objective</td>
<td>contact properly with the merchant</td>
</tr>
<tr>
<td>Buyer</td>
<td>Vision</td>
<td>find the perfect component that needed</td>
</tr>
</tbody>
</table>

Table 2-1 Example Buyer’s Ends

![Diagram](image)
### Table 2-2 Example Seller’s ends

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>provide customer with the best product and service</td>
<td>Goal</td>
<td>Expand the market, develop marketing strategies suited to the business, diversification of development (horizontal and vertical development)</td>
</tr>
<tr>
<td>developing objective</td>
<td>Objective</td>
<td></td>
</tr>
<tr>
<td>provide customer with the best product and service</td>
<td>Vision</td>
<td>in 3-4 years development, becomes one of the famous enterprise in related manufacturing domain.</td>
</tr>
<tr>
<td>market objective</td>
<td>Objective</td>
<td></td>
</tr>
<tr>
<td>production objective</td>
<td>Objective</td>
<td>reach 1000,000 productions per year by the end of this year</td>
</tr>
</tbody>
</table>

Figure 2-2 Sellers ends

### Table 2-3 Example Buyer’s means

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buyer</td>
<td>Strategy</td>
<td>medium ads</td>
</tr>
<tr>
<td>Buyer</td>
<td>Objective</td>
<td>find the proper merchant</td>
</tr>
<tr>
<td>Buyer</td>
<td>Strategy</td>
<td>invite public bidding</td>
</tr>
<tr>
<td>Buyer</td>
<td>Objective</td>
<td>contact properly with the merchant</td>
</tr>
<tr>
<td>Buyer</td>
<td>Objective</td>
<td>define a budget for the product</td>
</tr>
<tr>
<td>Buyer</td>
<td>Tactic</td>
<td>hold a formal meeting for the trade</td>
</tr>
<tr>
<td>Buyer</td>
<td>Tactic</td>
<td>market investigation</td>
</tr>
<tr>
<td>Buyer</td>
<td>Strategy</td>
<td>find a way to evaluate the price</td>
</tr>
</tbody>
</table>


Figure 2-3 Buyer’s means

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>developing objective</td>
<td>Objective</td>
<td>Expand the market, develop marketing strategies suited to business, diversification of development (horizontal and vertical development)</td>
</tr>
<tr>
<td>market objective</td>
<td>Objective</td>
<td>in 3-4 years development, becomes one of the famous enterprise in related manufacturing domain.</td>
</tr>
<tr>
<td>production objective</td>
<td>objective</td>
<td>reach 1000,000 production per year by the end of this year</td>
</tr>
<tr>
<td>provide customer with the best product and service</td>
<td>Goal</td>
<td></td>
</tr>
<tr>
<td>bench-mark strategy</td>
<td>Strategy</td>
<td>On the basis of the original brand, innovate, and hard work on services, make famous brands. make credible quality products for users.</td>
</tr>
<tr>
<td>production strategy</td>
<td>Strategy</td>
<td>based on existing product and service, expend production line both in horizontal and vertical dimension.</td>
</tr>
<tr>
<td>developing strategy</td>
<td>Strategy</td>
<td></td>
</tr>
<tr>
<td>market strategy</td>
<td>Strategy</td>
<td>Firmly grasp market share while at the same time, tries to broaden sales channels.</td>
</tr>
</tbody>
</table>
II.1.2 Possible Mapping between BMM and PIM4Agent

Business motivation model leads the analysis of business process and role definitions. We have known the core concepts of business motivation model are ends and means. There are other important concepts in BMM that relates with business process. Talking about this, we need to mention the gent oriented modelling. The core intention of business motivation model is “motivation”, which coherence with the Believe, Desire, Intention (BDI) concepts in Agent aspect.

The agent aspect describes single autonomous entity, the capabilities which they have to solve tasks and roles play within the Multi-Agent System (MAS). The agent aspect allows modeling cooperation between agents, either as loose collaboration or as a form of organization. It also defines the relationship between agents and roles they play in cooperation. Additionally, the meta-model defines to which resources (from its surrounding environment) and to which capabilities – a concept that groups behaviors – an agent has access to. The agent meta-model is depicted in Figure 2-5.
Cooperation represents the interaction between agents performing the required set of roles. The detailed realization of this interaction is described by a protocol that indicates what are the messages to be expected from each roles at each point in time. Organization illustrates the social structure agents can take part in. The Organization is a special kind of cooperation that also has the same characteristics of an agent. Therefore, the Organization can perform roles and have capabilities which can be performed by its members, be it agents or sub organizations. The multiple inheritance of the Organization, from the agent and the cooperation, also allows it to have its own internal protocol that specifies how the Organization coordinates its members.

Now we bring in other BMM concepts the following picture illustrate the influencer concept structure of BMM.

![Influencer concept structure of BMM](image)

An Influencing Organization is an organization that is external to the enterprise modeled in a given enterprise. An Influencing Organization is the source of Influencers. An Influencer may have multiple sources, or none. The model also supports Organization Category for categorization of Influencing Organizations, but does not provide a default set of categories. Enterprises can define whatever Organization Categories they need.

Influencers can be mapped to agent roles in the team-mode for the environment of the enterprise systems.

![Influencing Organizations of Buyer](image)
In Buyer_Seller example, the participants contain seller, buyer, order processor, the shipper, the product manufacture. From the role point of view these participant are organizations that can influencing each other.

And there exist actually influencing organizations in the environment, such as: government, standard organization, natural resources, some can be defined as resources (typically natural resources). Some can be defined as agents that can potentially or obviously influence the core role in the system.

The means corresponds with the Plan for the agent units; the mission is the boundaries for the agent scope definition. Each objectives for the ends can be defined as the exact conditions that the agent unit want to achieve, each means are the overall description for the process to acquire the conditions.

So I propose mapping rules as follows:
Rule 1: Means -> Plan

A Plan can be considered as a specialization of the abstract Behavior to specify an agent’s internal processes. A Plan contains a set of Flows and Activities. The Activities are linked to each other via Flows which are either of the type ControlFlow or InformationFlow. Furthermore, a Plan has a reference to a set of MessageFlows that are implemented by the Plan. This means that the internal behavior is defined that is necessary to guarantee the adequate message exchange in accordance to particular Protocol the MessageFlow belongs to.

A Means represents any device, capability, regime, technique, restriction, agency, instrument, or method that may be called upon, activated, or enforced to achieve Ends. Remember that a Means does not indicate either the steps (business processes and workflow) necessary to exploit it, nor responsibility for such tasks, but rather only the capabilities that can be exploited to achieve the desired Ends.

For the analysis of plan, we need means to ends analysis for the business motivation.
Rule 2: InfluencingOrganization ->Organization

As figure 2-4 shows the influence analysis for the Buyer_Seller example. A business process model is required for describing the behavior of the activities inside each organization. The OMG organization’s UML diagrams contain activities for describing the business Processing Model. But here we have another choice – the Business Process Model for the business process modeling in the next chapter.
II.2 BPMN - The Business Process Modeling Notation

Business process management (BPM) is a method of efficiently aligning an organization with the wants and needs of clients. It is a holistic management approach that promotes business effectiveness and efficiency while striving for innovation, flexibility and integration with technology. As organizations strive for attainment of their objectives, BPM attempts to continuously improve processes - the process to define, measure and improve your processes – a 'process optimization' process.

II.2.1 BPMN example for Buyer_Seller

Here we would simulate the process of Buyer & Seller Example. Buyer would purchase products through online shop, which is Seller, doing the online payment and offline products shipping.

II.2.1.1 Top View Description

This top view of BPMN (see Figure 2-9) and the detail description are all built by Eclipse BPMN plug-in. This Buyer_Seller Example describes the process which Buyer purchases products from an online shop, which is Seller. The top view is more general, just to describe communication between different participants. In this diagram, it contains totally four pools, which mapping four participants in Service model:

- Buyer: User who will come to online shop to buy products.
- Seller: An online shop, which faces buyer and being the mediator to collaborate with bank and logistic company, which is shipper.
- Bank: Processing the payment function during the process.
- Shipper: Processing the shipping function during the process, the delegate would be logistic company.

![Figure 2-9 Top view of BPMN](image-url)
In this top view diagram, the main entity would be task. Executing Sequence would follow the message flow and work flow. Buyer would be the default starter. Main flow is:

- Communication between Buyer and Seller: processing products’ order and shopping cart.
- Communication between Seller and Bank: processing set of payment functions.
- Communication between Seller and Shipper: processing set of production shipping.

II.2.1.2 Detail View Description

When Buyer login the online shop platform, he will first choose products (see Figure 2-10). This process will communicate with Seller three times to make sure the products. The loop set demonstrate user can choose products several times.

Figure 2-10 Choose Products

Figure 2-11 simulates how to search and return a product list. The different height of tasks show the different lanes in the shop pool. The highest level is the GUI level in PSM view. The middle level stands for controller and calculate level. Bottom level would be the database level.

Figure 2-11 Search Products
Figure 2-12 shows online shop regulate shopping cart after each time’s product order. It is also departed into different lanes. The higher lanes is also the GUI which user could see from web pages. The initiate selection would be inter-control inside shop.

![Figure 2-12 Set Shopping Cart](image)

Figure 2-12 Set Shopping Cart

Figure 2-13 shows when buyer decide to buy products with shopping cart, and receive the return confirmation. First user send a message event to show online shop that he want to buy the products. After sets of operations, system tell user the return information. Then buyer could decide he will really buy it or not.

![Figure 2-13 Buy Products](image)

Figure 2-13 Buy Products
Figure 2-14 shows how shop inter-execute after they know user want to buy it. First they will initiate total payment, if user’s account is not enough for payment, there should be a compensate task that the product in shopping cart should release, then other buyer could choose these products. After this is the normal progress, got the information and communicate with bank. Especially, this combines “set payment” and “apply payment” together.

After bank receives the information transfer from shop, bank will contact with customer, let user decide if they would pay for these products. Then, bank will execute the money transfer flow. (See figure 2-15)

After payment, shop will inform shipper, such as logistic company, ship the products to buyer. Shipper would process transportation and get the result back. (See figure 2-16)
II.2.2 Mapping between BPMN and UPMS

Currently many different modeling languages are used for workflow definitions in BPM systems. BPMN, as we mentioned, provide an easy readable graphical notation for workflow processes. The elements were chosen to be distinguishable from each other and to utilize shapes that are familiar to most modelers. For example, activities are rectangles but decisions are diamonds. Until now, several world-wide huge IT company, such as IBM and SAP, are also utilizing BPMN to help them handling business description.

While The emerging UPMS (UML Profile and Metamodel) standard from OMG aims to address the need for describing the interactions among the participants in a service oriented architecture independent of technological realizations. The UPMS model should also bridge the gap between the business analysts and the IT developers and support mapping and alignment between enterprise and IT models.

But still, we can find some conceptual difference between BPMN and UPMS. Here gives a short advice example describing how to mapping BPMN and UPMS together.

II.2.2.1 BPMN pre-work

We can see that there are some special restriction for our BPMN example comparing other examples. In our design, we could give two BPMN designs. One is called top view and another called detail view which are all mentioned before. They have their independent functionality and responsibilities.

BPMN Top view is the structure of detail view. It gives the main activities of the whole business process. In order to mapping this efficiently to the UPMS, we should make some rule for the BPMN top view:

- The only elements appears in BPMN top view are activities, message flows, and begin & end events.
- The activities, most of them, are directly mapping to the BPMN detail view. Or we can say that, BPMN detail view is the extension of BPMN top view.
- All the activities appear in BPMN top view should all have message flows connecting each other. The ones with no message flows connected would be considered redundantly.

The reason why we set these rules are according to the mapping rules which we could introduce it later. The rules for BPMN detail view are:

- As the detail view is the extension of top view, it’s just enlarging the content, not revising it. So we could just using sub-process to detail describe what’s inside the activities.
- Each sub-process have a start event and an end event. There also could be message flow coming from other sub-process’ certain activity. But they should all follows the top view, be consistent with the top view.
II.2.2.2 Mapping rules

As the pre-work has been done, it’s easy to describe the rules to mapping BPMN and UPMS. We all know that the main elements in UPMS are: participant, port, interface, operations. Others are not so important as these things. So our goals becomes how to mapping elements from different layers, different models, different point of view appropriately.

First is the Participant. Participants are types of actors defined both by the roles they play in services architectures (the participant role) and the “contract” requirements of entities playing those roles. While in BPMN, A Pool represents a Participant in a Process. It is also acts as a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations. So the a Pool should be directly mapping to a Participant.

Second is the Port and Interface. We can see that there are two kinds of interfaces supported by the port: required interfaces and provided interfaces. The original port’s definition comes from collaboration. The purpose of a collaboration is to show how kinds of entities work together for some purpose. Collaborations are based on the concepts of roles. A role defines how entities are involved in that collaboration. We use port to fulfill the roles. Well, service interface is the super type of “Provider” and “Consumer” and is used for service contracts with more than two roles. The activity from different roles fulfils the function. They use message flow to combine each other and work together, just like collaboration. So the activities should be the port. The interface, theoretically, has no direct mapping elements from BPMN as the functions point of view. Whileing we could just using the meaning of the interface to give it’s name. such as the two ports are process payment and confirm payment, their interface name should be processing payment and confirming payment. While most of the time, the two interface should be the same name to show that they are combining with each other. So mostly, we just use the require interface’s name.

As the operation inside the interface, we have to see the BPMN detail view. The activities inside the sub-process would be the operations inside the interface. Which pools the activities come from, which participant and port it should be corresponding with.

Last tip, the business process always having the go and back, which means there are message flows goes out and other message flows returns. But there are some situations that when the message flows are sent out, there would not be direct message flow come back because other pools is process other things during this period. So we have to receive the message flow after several other activities. In that case, we have to find them, which is not hard, and tackle them together like one activities.

II.2.2.3 Mapping implementation

Since we talked about the mapping rules, we could take the mapping step by step in order to show the mapping steps. Here we choose the Papyrus as the UPMS tool. The reasons are:

- Open sources
- XML Code supported

As using papyrus is not essential here, after we built and import the profile for UPMS, we only give the components and realization classes. Figure 2-17 shows the customer component and realization classes.
Here we can contrast with BPMN diagram’s customer part. In BPMN top view, such as Figure 2-18 which is BPMN Customer top view. There are three activities: choose products, buy products and confirm payment. And you can see the realization classes: choose products, buy products, confirm payment and confirm shipping which have relationship with the interfaces they would using or realize.

Figure 2-18 BPMN customer top view

So, the whole component and realization classes diagram would all built like:

- Taking the BPMN top view
- Find each pools and it’s activities
- Draw the component and realization classes diagram according to the message flow between activities

Totally, there are four pools in the BPMN diagram, so there are four component and realization classes diagram. They are figure 2-19, figure 2-20 and figure 2-21.
Figure 2-19 Shop component and realization classes

Figure 2-20 Bank component and realization classes

Figure 2-21 Transport component and realization classes
Now we should built the component diagram. But the technology in using Papyrus will not be focusing on this chapter. We just look at the final diagram. But the elements in this component diagram which fulfils the requirement of UPMS are all comes from the diagram we mentioned above. Figure 2-22 is the final UPMS service diagram.

![UPMS service diagram](image)

Figure 2-22 UPMS service diagram

The elements in UPMS service diagram are all generated from the before component and realization classes. What we should do is connecting the required interface and provided interface. But as the operation inside the interface, Papyrus didn’t support that function to fill things inside. But we should know what’s in it and how to map those. Take the Choose Product port as example, we have mentioned the diagram in Figure 2-23.

![Choose Products](image)

Figure 2-23 Choose Products

So the operations inside the choosing product interface should consist operation: view products, receiving product list and select products. As this regulation, we could build the table that the operations inside the interfaces. Such as Table 2-5
Supplement on Buyer/Seller Purchase Order Example

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Operations</th>
</tr>
</thead>
</table>
| Choosing product | View products  
Receiving product list  
Select products  
Receiving keywords  
Search product  
Find product list  
Obtain product information  
Generate product list  
Show product list  
Change order  
Initiate order  
Show shopping cart |
| Buying product | Buy product  
Receive confirmation  
Send & receive confirmation or deny  
Initiate payment  
Generate payment confirmation |
| Applying & returning payment result | Apply payment  
Receiving payment  
Returning payment result |
| Processing payment | Receiving confirmation  
Sent & receive payment info  
Sent & receive payment confirmation  
Execute payment |
| Applying & returning transportation result | Initiate & send transportation info  
Receiving transportation info  
Returning transportation result |
| Processing transportation | Initiate transportation plan  
Execute transportation  
Receive products  
Send & receive confirmation  
Send confirmation to shop |

Table 2-5 operations inside the interfaces

In fact, these operations’ names are not so suitable as the service level. We could and should change the information which according to the IT programming facilities and culture.

II.3 COMET Business model overview

II.3.1 COMET Business model overview

The business model is used to express the part played by the product (system or component) being developed in the context of the business that will fund its development (or purchase it) and use it. It also provides elements that link extremely well to the architecture model.

The business model includes goals, business processes, steps within business processes, roles and resources. The scope (or domain) of the model is any part of the world defined as interesting for a company, organization or others, and which has some impact on the required behavior or other characteristic of the product. The business model might be broadly or narrowly scoped, e.g. describing the entire business of a company or describing the immediate environment and context of a product under consideration. Thus, the business model is recursive in the sense that some business models might detail aspects of another more broadly scoped/higher level business model.

The set of work products of a business model is outlined below.
II.3.2 The business resource model – UML Class model and OWL

II.3.2.1 Modelling Objectives

The Business Resource Model is an information model that identifies and defines the main things (and concepts) of the domain that are relevant to the Product, these being, in general, those things that do things in the business (including the Product itself), and those things that have things done to or with them, and details the relationships between these concepts.

The Business Resource Model can be specified as an information model using UML class diagrams, it can be linked to an ontology model, for instance expressed in OWL (or a UML profile for OWL) using semantic annotation, or be specified directly as an OWL model.

A Product comprises a complete and consistent set of models for the Product, namely Business Model, Requirements Model, Architecture Model and one or more Platform Specific Model together with the corresponding Executables.

II.3.2.2 Deliverables and notation

The Business Resource Model takes the form of UML class model as part of the PIM for the Product (see the example: model structure). In cases where it is useful, it may be accompanied by state machine models for any of the resources modeled.

The Figure 2-24 below is a summary of the order process of the Buyer_Seller example. The customer will search through available products in the web shop. Then decide to add an order of available products. When the customer finished, he will confirm the order and sent order to the web shop. The order will then have status unconfirmed and the web shop will instantiate the process for retrieving transport proposals and the prices for these. Alternative transport plans and prices are given to customer and he has to decide which plan fit his needs, if any. When a customer picks the plan, he submits his choice with payment information to the web shop. Web shop then processes the payment information and if payment check is confirmed, a shipping request would be sent to shipping company and the buyer gets a confirmation that order is completed.

Figure 2-24 Order process of the Buyer_Seller example
II.3.2.3 PIM for the Buyer and Seller

PIM for Buyer

The Figure 2-25 is the Platform-independent model of buyer. There are totally seven entities: Buyer, Order, Orderline, Lineitem, Item, InvoiceLine and Invoice.

PIM for Seller

The Figure 2-26 is the Platform-independent model of seller. There are seven entities: Customer, PricedDocument, PurchaseOrderline, PurchaseOrder, Item, InvoiceLine and Invoice in it.
II.4 Work analysis refinement model (WARM)

II.4.1 Modeling objectives

As its name suggests, the WARM is a refinement of the basic Business model which concentrates on "Work Analysis", i.e., which kinds of resources do which kind of work. Specifically, it concentrates on those resources that will form part of the Product being developed, and the behavior (expressed as Steps in Processes) that they will be required to exhibit.
II.4.2 Methods and techniques

The WARM is not a separate model, or even a separately identifiable element in a model. It is a final refinement of both the Business Process & Roles model and the Business Resource model, (both, themselves, distributed amongst a variety of Community models), in which the Steps in the Business Process model are refined and assigned to specific refined Actor Resources. It is from this part of the model that the role, in the business, of a particular IT element, can be ascertained.

The WARM concepts that specialize Actor Resource are illustrated in the figure below:

Resources are refined to identify the IT elements that fulfill roles in the business, as follows:

- A Product in the WARM is an actor resource that fulfils a role in the business and maps to the executable part of a Product.
- A Business Service is a Resource with a role in the business, and represents an IT element that maps to a Business Service Component in the corresponding Architecture model.
- A Workflow is an actor resource that maps to a Workflow Definition in a corresponding Architecture model which contains the business logic of a process and determines the sequencing of steps within any instance of that process.
- A Human/Tool is an actor resource with a role in the business that is fulfilled by a human working with an IT element that maps to a Tool Component in the corresponding Architecture model.
- An Other System is an actor resource that represents an external automatic system that fulfils a role in the Business Process Model.
- In addition, so that the model can be complete, we identify:
  - A Human (user) is an actor resource that represents a human that fulfils a role in the Business Process Model.
  - The System represents the complete IT facility that supports the business including all Products and the Infrastructure.

The WARM concepts that specialize the business process modeling concept of Step are illustrated below:
In WARM refinement of the Business Process model, the kinds of step performed by resources in the model are further categorized as follows:

- A Human Step is a step performed by a human with no involvement of the Product being modeled.
- An Extended Step is a step in which the intermediate states are of interest to the business, and may have to be remembered. This may be either because there are business reasons for such interest, or because other factors, including technology, require that the business be concerned. An extended step is a candidate for choreography by a Workflow.
- A Tool Step is a step performed by a human user interacting with a tool that is part of the Product. The human user will use some form of interactive device (e.g. a GUI) to interact with the Product. A Tool Step is a candidate for realizations by a Tool Component.
- An Immediate Step is a step that is required to complete as soon as possible, and whose intermediate states are of no concern to the business. It is performed autonomously, with no intervention from a human. An Immediate Step may be mapped to an Operation on a Business Service Component (Process) in the Architecture Model. Each such operation on a Business Service Component in the Architecture will run in an ACID transaction, thereby either completing or leaving state as it was. The Name of the step is the verb or verb phrase that describes the process (for example, "Modify Customer record"). If a resource is specified as part of the name or description (for example "Modify Customer using Info from Sales Person") then the model should identify the relevant Resources.

II.4.3 WARM implementation

As the WARM is not a separate model, but an extension to and refinement of the Business Process model and the Business Resource model, here we take BPMN diagram as an example.
II.4.3.1 WARM BPMN Implementation

According to the WARM method, we set keywords to each activities in order to divide the business activities’ types. Here we take three main representative “WARM” types: Human, Tool and Immediate.

- **Human type** represents the activities that are all processed by person, with no involvement of the computer’s help
- **Tool type** represents the activities that interacting with a tool by a human. In another word, the activities would be controlled by human, but just by handling the IT infrastructure
- **Immediate type** represents the activities that performed autonomously, with no intervention from a human. Most of the time, it’s handled by automated computer and should finish the work as soon as possible.

Figure 2-29 gives an example that how to utilize WARM into the BPMN diagram. This process is quite easy, we should give a stereotype or a keyword to identify the activity’s type. Here we just add a keyword at the beginning of the name. This process should always been done by designing process period.

![BPMN WARM implementation diagram](image)

II.4.3.2 WARM UPMS Implementation

As we have settled the mapping between BPMN and UPMS, we know that there are direct relationship between the BPMN activities and UPMS elements. So after we use WARM method to refine the type of activities, we could also use WARM to settle the type for the UPMS elements, which is port. Figure 2-30 is the example of using WARM method to refine the UPMS diagram.
Figure 2-30 UMPS WARM Diagram
III COMET - Requirements model

III.1 Introduction – COMET Requirements model

The requirements model identifies the system requirements. These include functional requirements, non-functional requirements (quality of service) and constraints.

III.2 COMET - Requirements model to service architecture model transformation

III.2.1 Requirements model to architecture model transformation

We see a possibility of transforming the requirements model to a first draft of the architecture model, for example by generating a UserService for each actor (which we see is quite common the case). However, the transformation is a one-way transformation from requirement model to architecture model. It will constrain the models unnecessarily, especially the requirement model, to maintain a round trip transformation between the requirements model and the architecture model. The following transformations are possible:

Map each actor to a UserService and create an Interface as a provided interface of the actor. For each of the use cases that the actor relates to add a corresponding operation to the Interface. Naming conventions: Use the Actor name to name the UserService and the same for the interface prefixed with an “I”. The operations should be named according to the use cases.

Map each <<Manage>> use case to a BusinessService providing CRUD operations as well as find and collection operations for the Resource(s) that are related to the <<Manage>> use case. Naming conventions: use the resource name prefixed with “Manage”

Possible way to handle extends and includes relationships:
Include -> reusable UserService with interface providing the “include” service or an operation in the interface of the extended UserService.
Extends-> Operation in the interface of the extended UserService

III.2.1.1 Transformation specification

MOF have defined a metamodel for model transformations below we have specified the transformation rules for the Actor to UserService mapping based on this scheme. Figure 19 describes a simple transformation from a UML use case model to an architecture model with UserService components (classifiers) with interfaces.
The transformation map ‘Actor-2-UserServiceMap’ has one type of source object, an Actor. This signifies that the extent of actors in the source model is input for this transformation (one at a time). Source and target objects are modeled in terms of associations to types from the source and target metamodels, respectively. Source and target associations can be named, which signify named references to the source/target objects. A target association signifies the creation of one instance of the target type. From the example, the target ‘UserService’ signifies a creation of a UserService object. The references to these objects have global scope and can be used in, for example, FeatureMaps in the specification.

The classifier maps are either part of the transformation map or other classifier maps, defined through aggregate associations. An association defines a transformation path that implicitly (or explicitly, through OCL statements) carries source objects. When no constraint is specified, the sources of the owner become sources for the contained element (classifier map).

Figure 3-2 depicts a detailing of the transformation, which includes simple feature maps. A feature map is modeled as an attribute and may contain assignments or action specifications. An assignment typically maps basic properties of the source, to basic properties of the target, e.g. their names. An action specification is a more complex operation that e.g. adds a realization dependency to a target object.
OCL constraints are used to constrain the input domain for a transformation class, based on the context of the owner. Typically, these define selection criteria for the set of objects to be handled by the receiving transformation class. OCL constraints are assigned to the associations between transformation classes. In the example in Figure 21, the OCL constraint on the association selects all use cases an actor communicates with.

![Figure 3-3 OCL constraints](image)

In addition to transforming simple properties within FeatureMaps, more complex operations are needed to transform relational features, like adding an operation to an interface. Two different methods are used to support this. The first one is implicitly performed, by adding properties by referring relations from the metamodel, as in Figure 3-4.

![Figure 3-4 Referring to metamodel relations](image)

The second way is to specify feature maps that add a relationship to the target object, as shown in Figure 3-5.

![Figure 3-5 Adding properties in a FeatureMap](image)
The feature map accesses the properties of the target object and imposes an addition of a feature. The add operation is assumed to be handled by the target object.

III.3 Requirements model example

III.3.1 Product Vision and Description

III.3.1.1 Vision
The web shop manage tool is a web-based and highly automated tool to support the process and information around management of the order in the shop. It’s main task will be:
- Orders leads and surveys.
- Send automatic order list on changes and conflicts.
- Information about current resource usage and potential backlog.

III.3.1.2 Description
The web shop manage tool will help to administer the utilization of the production. It will mainly benefit three user groups; Buyer, Seller and management.
- Buyer will use it to put a production or service onto a order. The web shop manage tool will give an overview over the current storage and also which production qualifies for the order.
- Seller will make purchase suggestions to management, which then need to be confirmed by the global marine management.
- For seller the web shop manage tool should work as a planning aid. It will automatically warn about changes and it will have detailed production information available on-line.
- Management will be able to use the web shop manage tool to assess the current resource usage, confirm orders and to reschedule storage list.

The web shop manage tool will be web based and automatically updated with the current progress rates from Introspection. A consequence of this approach is that there needs to be one global master order. In order to visualize alternative scenarios each user can create private purchase order.

III.3.2 Introduction
This chapter specifies the requirements and high-level use cases for the web shop manage tool. The web shop manage tool is a complete redesign of the purchase order list. We tried to design it as a 'light' tool, without too much intelligence. The advantage of this design is that we can deliver the web shop manage tool fairly fast. We have the luxury of owning a good old version.

III.3.3 Service information model – Class Diagrams

III.3.3.1 Use Cases
With the presentations of use cases in this chapter, we want to follow a quite widely followed model on how to engineer a project. Use cases are part of a structured approach to capture all interactions which a system should do and at the same time define all interacting user groups and systems.

The use cases describe how actors interact with the system. Actors can be users or other systems, which normally extract or import information. In this chapter a high level view of the use cases is presented. The main aim at this point is to assure that all use cases have been found and all actors were identified.
III.3.3.2 System Boundary model

Figure 3-6 Subsystem grouping use cases

III.3.3.3 Definition of actors

- Seller: The seller department is responsible for searching product in the storage, get the product information for Buyer, receive the product list and confirmation form Buyer, receive the payment from Bank and return the payment result to the Bank, receive the ship information from shipper.

- Buyer: Buyer will be responsible for the view production on the shop list and get the product information from Seller, select the product in the product list which is given by Seller, initiate or change order, buy product, initiate payment and receive reply from Seller, receive product from Shipper and send the confirmation to Shipper, send payment information to Bank, receive payment confirmation from Bank.
Bank: Bank will be responsible to receive payment information, send payment confirmation and execute the payment with Buyer and apply payment with Seller.

Shipper: Shipper will be responsible for initiating shipping plan, execute ship and send confirmation to Buyer, initiate and ship information, returning ship result to Seller.

### III.3.3.4 Summary list of use cases

1. View products
2. Receiving product list
3. Select products
4. Search product
5. Find product list
6. Obtain product information
7. Generate product list
8. Show product list
9. Change order
10. Initiate order
11. Buy product
12. Receive confirmation
13. Send & receive confirmation or deny
14. Initiate payment
15. Generate payment confirmation
16. Apply payment
17. Receiving payment
18. Returning payment result
19. Receiving confirmation
20. Sent & receive payment info
21. Sent & receive payment confirmation
22. Execute payment
23. Initiate & send transportation info
24. Receiving transportation info
25. Returning transportation result
26. Initiate transportation plan
27. Execute transportation
28. Receive products
29. Send & receive confirmation
30. Send confirmation to shop
IV COMET-S - Service architecture model (UPMS)

IV.1 Introduction

Service Oriented Architecture (SOA) is a way of organizing and understanding organizations, communities and systems to maximize agility, scale and interoperability. The SOA approach is simple – people, organizations and systems provide services to each other. These services allow us to get something done without doing it ourselves or even without knowing how to do it - this provides us with efficiency and agility. Services also enable us to offer our capabilities to others in exchange for some value – thus establishing a community, process or marketplace. The SOA paradigm works equally well for integrating existing capabilities as well as creating and integrating new capabilities.

A service is a capability offered through a well-defined interface and available to a community (which may be the general public) using a standard communications infrastructure and communication protocols. SOA is an architectural paradigm for defining how people, organizations and systems provide and use services to achieve results. SOA-Pro provides a standard way to architect and model SOA solutions using the Unified Modeling Language, UML. The profile uses the built-in extension mechanisms of UML to define SOA concepts in terms of existing UML concepts. SOA-Pro can be used with current “off the shelf” UML tools but some tools may offer enhanced, SOA specific, support.

IV.1.1 Service Profile and meta-model

Figure 4-1 Latest Diagram SOA Profile
Supplement on Buyer/Seller Purchase Order Example

Figure 4-2 Service diagram

Services Diagram provides an overview of the SOA-Pro meta-model. Package Services defines a new set of capabilities that can be merged with UML2 to extend it with service modeling capabilities. Packages Partitions and Policies extend Services with some additional capabilities. The various integration packages extend Services with integrations with other OMG specifications. These integrations are done using package merge to weave in possibly optional service modeling aspects and capabilities.

The primary purpose of an SOA is to provide an effective means of connecting consumers with needs to providers with appropriately matching capabilities. It is these participants and their connections that make up the services architecture.

IV.2 Buyer_Seller examples

The Figure 4-3 shows service participants, “"Buyer” that has <<service>> port named “ConfirmPayment”. The type of the service port is the UML interface “ConfirmPayment”. “Buyer” has the <<requisition>> port named “Choose Product” that requires the service of “choosing product”. Note that a participant may also offer other services on other service ports.
Where a basic service is more complex than can be expressed in a UML Interface, a <<Service Interface>> can be defined as shown below.
Like a UML interface, a <<Service Interface>> can be the type of a service port. The service interface has the additional features that it can specify a bi-directional service – where both the provider and consumer have responsibilities to send and receive messages and events. The <<Service Interface>> is defined from the perspective of the service provider using three primary sections: The interfaces, the Service Interface class and the Behavior.

- **The Interfaces** are standard UML interfaces that are provided or used by the service interface. The interface that is provided specifies the messages that will be received by the provider (and correspondingly sent by the consumer). The interface that is <used> by the service interface defines messages or events that will be received by the consumer (and correspondingly sent by the provider).

- **The Service Interface and enclosed parts** specify the roles that will be played by the parties involved with the service. The role that is typed by the provided interface will be played by the service provider. The role that is typed by the <<used>> interface will be played by the consumer.

- **The behavior** specifies the interactions between the provider and consumer – the contract of interaction, without specifying how either party implements their role. Any UML behavior specification can be used, but interaction diagrams are the most common.

**IV.2.1.1 Participants.**

As represented in Figure 4-8, buyer, seller, shipper and bank are service participants. Participants are types of actors defined both by the roles they play in services architectures and the services they provide and use. Participants can represent software components, organizations, actors or individuals. Services are provided and used through having service ports that have a type compatible with the Service Interface, or UML Interface they must provide and use.

These interaction points use the UML concept of a “port”. A port is the interaction point where a classifier interacts with other classifiers. A port typed by a service interface, is a service port.

![Figure 4-5](image)

A service port is the point of interaction on a participant where a service is actually provided or consumed. On a service provider this can be thought of as the “offer” of the service (based on the service interface), the service port is the point of interaction for engaging participants in a service via its service interfaces.
Using a service is also defined using a port stereotyped as a \textit{<<Requisition>>}. The type of a requisition port is also a Service Interface or UML Interface, as it is with a service port. The requisition port is the conjugate of a service port in that it defined the use of a service. As we will see below this will allow us to connect service providers and consumers in a component.

The Shipper participant example below shows that Shipper provides the “Shipping” service using the “receive shipping” service interface defines on the “receive and return Shipping Result” service port and uses the “Processing shipping” service defined as a \textit{<<Requisition>>} of the ShippingRequest interface. Note that this requisition is the conjugate of the service which is offered by a shipper, as defined below.

By using service and requisition ports SOA-Pro can define both the service responsibilities and requirements of participants at the business or technical level.
IV.2.1.2 Service contract.

A key part of a service is the Service Contract, see Figure 4-9. The service contract defines the terms, conditions, interfaces and choreography that interacting participants must agree to (directly or indirectly) for the service to be enacted – the service contract is the full specification of a service which includes all the information, choreography and any other “terms and conditions” of the service. The service contract is binding on both the provider and consumer of that service. The basis of the service contract is also a UML collaboration that is focused on the interactions involved in providing a service. A participant plays a role in the larger scope of a service architecture and also plays a role as the provider or user of service contracts.

There is also a Service Interface for each of the involved parties in a service contract. An important part of the Service Contract is the choreography. The Choreography is a specification of what is transmitted and when it is transmitted between parties to enact a service. The choreography specifies exchanges between the parties – the data, assets and obligations that go between the service provider, consumer and any other service interface.
IV.3 COMET - Service architecture metamodel

- The COMBINE 4+2 Tier Reference Architecture
  - Component Types
    - Tier Components
    - Composite Components
    - Workflow Components
  - Type Libraries
    - Examples
V  COMET-S - Platform specific model

V.1  Introduction
A platform-specific model is a model of a software or business system that is linked to a specific technological platform. Platform-specific models are indispensable for the actual implementation of a system. As the name indicates this model defines the result of mapping the service model to an implementation on a particular infrastructure.

V.2  Platform profile model – JEE Meta-model

V.2.1  Goals
The purpose of this model is to specify the system in alignment to the actual technology profile for the specific platform (e.g. based on the UML profile for EJB).
VI  JEE

VI.1  JEE Meta-model

VI.1.1  Introduction

The Java EE application model begins with the Java programming language and the Java virtual machine. The proven portability, security, and developer productivity they provide forms the basis of the application model. Java EE is designed to support applications that implement enterprise services for customers, employees, suppliers, partners, and others who make demands on or contributions to the enterprise. Such applications are inherently complex, potentially accessing data from a variety of sources and distributing applications to a variety of clients.

The business functions to support these various users are conducted in the middle tier. The middle tier represents an environment that is closely controlled by an enterprise’s information technology department.

The Java EE application model defines architecture for implementing services as multitier applications that deliver the scalability, accessibility, and manageability needed by enterprise-level applications. This model partitions the work needed to implement a multitier service into two parts: the business and presentation logic to be implemented by the developer, and the standard system services provided by the JEE.

VI.1.2  Distributed Multi-tiered Applications

The Java EE platform uses a distributed multi-tiered application model for enterprise applications. Application logic is divided into components according to function, and the various application components that make up a Java EE application are installed on different machines depending on the tier in the multi-tiered Java EE environment to which the application component belongs.

Figure 6-1 show two multi-tiered Java EE applications divided into the tiers described in the following list.

- Client-tier components run on the client machine.
- Web-tier components run on the Java EE server.
- Business-tier components run on the Java EE server.
- Enterprise information system (EIS)-tier software runs on the EIS server.

![Figure 6-1 Multitiered Applications](image-url)
VI.1.3 Java EE Components

Java EE applications are made up of components. A Java EE component is a self-contained functional software unit that is assembled into a Java EE application with its related classes and files and that communicates with other components. The Java EE specification defines the following Java EE components:

- Application clients and applets are components that run on the client.
- Java Servlet, Java Server Faces, and Java Server Pages TM (JSPTM) technology components are web components that run on the server.
- Enterprise JavaBeansTM (EJBTM) components (enterprise beans) are business components that run on the server.

Java EE components are written in the Java programming language and are compiled in the same way as any program in the language.

VI.1.4 Business Components

Business code, which is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in the business tier. Figure 6-2 shows how an enterprise bean receives data from client programs, processes it and sends it to the enterprise information system tier for storage. An enterprise bean also retrieves data from storage, processes it and sends it back to the client program.

VI.1.5 Enterprise Information System Tier

The enterprise information system tier handles EIS software and includes enterprise infrastructure systems such as enterprise resource planning (ERP), mainframe transaction processing, database systems, and other legacy information systems. For example, Java EE application components might need access to enterprise information systems for database connectivity.
VI.2 JEE Notation and UML Profile

Figure 6-3 illustrates the availability of the Java EE 5 platform APIs in each Java EE container type. The following sections give a brief summary of the technologies required by the Java EE platform, and the APIs used in Java EE applications.

VI.2.1 Enterprise JavaBeans Technology

An Enterprise JavaBeans (EJB) component, or enterprise bean, is a body of code having fields and methods to implement modules of business logic. Think of an enterprise bean as a building block that can be used alone or with other enterprise beans to execute business logic on the Java EE server.

There are two kinds of enterprise beans: session beans and message-driven beans. A session bean represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone. A message-driven bean combines features of a session bean and a message listener, allowing a business component to receive messages asynchronously.

In Java EE 5, entity beans have been replaced by Java persistence API entities. An entity represents persistent data stored in one row of a database table.

VI.2.2 Java Server Pages Technology

Java Server Pages (JSP) technology lets you put snippets of servlet code directly into a text-based document. A JSP page is a text-based document that contains two types of text: static data and JSP elements, which determine how the page constructs dynamic content.

VI.2.3 Java Server Faces

VI.2.3.1 Java Server Faces technology is a user interface framework for building web applications.

The main components of Java Server Faces technology are as follows:

- A GUI component framework.
- A flexible model for rendering components in different kinds of HTML or different markup languages and technologies. A Renderer object generates the markup to render the component and converts the data stored in a model object to types that can be represented in a view.
- A standard Render Kit for generating HTML/4.01 markup.

Figure 6-3 Java EE Platform APIs
VI.2.3.2 The following features support the GUI components:
- Input validation
- Event handling
- Data conversion between model objects and components
- Managed model object creation
- Page navigation configuration

VI.2.4 Java Transaction API

The Java Transaction API (JTA) provides a standard interface for demarcating transactions. The Java EE architecture provides a default auto commit to handle transaction commits and rollbacks. An auto commit means that any other applications that are viewing data will see the updated data after each database read or write operation. If your application performs two separate database access operations that depend on each other, you will want to use the JTA API to demarcate where the entire transaction, including both operations, begins, rolls back, and commits.

VI.2.5 Java Naming and Directory Interface

The Java Naming and Directory Interface (JNDI) provide naming and directory functionality, enabling applications to access multiple naming and directory services. It provides applications with methods for performing standard directory operations. Java EE naming services provide application clients, enterprise beans, and web components with access to a JNDI naming environment. A container implements the component’s environment and provides it to the component as a JNDI naming context.

A Java EE component can locate its environment naming context using JNDI interfaces. A component’s naming environment is stored directly in the environment naming context or in any of its direct or indirect sub-contexts. It can access named system-provided and user-defined objects. The names of system-provided objects are stored in the environment naming context, java:comp/env. JEE platform allows a component to name user-defined objects. A component should be named within a sub-context of the naming environment according to the type of the object.

VI.2.6 Simplified Systems Integration

The Java EE platform is a platform-independent, full systems integration solution that creates an open marketplace in which every vendor can sell to every customer. Such a marketplace encourages vendors to compete, not by trying to lock customers into their technologies but instead by trying to outdo each other in providing products and services that benefit customers. The Java EE 5 APIs enable systems and applications integration through:
- Unified application model across tiers with enterprise beans
- Simplified request-and-response mechanism with JSP pages and servlets
- Reliable security model with JAAS
- XML-based data interchange integration with JAXP, SAAJ, and JAX-WS
- Simplified interoperability with the J2EE Connector architecture
- Easy database connectivity with the JDBC API
- Enterprise application integration with message-driven beans, JMS, JTA, JNDI.
VI.3 JEE Explanation and Description

VI.3.1 Overview Introduction

Service-oriented architectures (SOA) promise to implement composite applications that offer location transparency and segregation of business logic. Location transparency allows consumers and providers of services to exchange messages without reference to one another’s concrete location. Segregation of business logic isolates the core processes of the application from other service providers and consumers.

In the loan application use case, a user applies online for a loan by filling out a loan request with necessary financial and personal information. The application’s business logic verifies the user information and accepts or rejects the loan request. The use case is summarized in Figure 6-4.

![Figure 6-4 Business Use Case Flow Chart for Example Composite Application](image)

VI.3.2 JAX-WS 2.0

JAX-WS 2.0 replaces an older API, JAX-RPC 1.1, extending it in many areas. The new specification supports multiple protocols, such as SOAP1.1, SOAP 1.2, and XML. JAX-WS uses JAXB 2.0 as its data binding model, relying on usage annotations to considerably simplify web service development. It also uses many annotations defined by the specification Web Services Metadata for the Java Platform and introduces steps to plug in multiple protocols instead of HTTP only.

VI.3.3 Java EE Web Service Architecture

Web service architecture, in general, allows a service to be defined abstractly, implemented, published and discovered, and used interoperable. You can decouple a web service implementation from its use by a client in a variety of ways-in programming model, logic, and transport. As a consequence, a web service that has been developed with the .NET platform can be used by a Java EE application, and vice versa.

In simplest terms, a service instance, called a Port component, is created and managed by a Java EE container, which in turn can be accessed by the client application. The Port component can also be referenced in client, web, and EJB containers.
VI.4 JEE Example for Purchaser Order

VI.4.1 Common Code of PurchaseOrder

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="javaeews"
nsURI="http://javaeews" nsPrefix="javaeews">
<eClassifiers xsi:type="ecore:EClass" name="NamedElement">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="Package" eSuperTypes="#//NamedElement">
  <eStructuralFeatures xsi:type="ecore:EReference" name="services" upperBound="-1" eType="#//WebService" containment="true"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="WebService" eSuperTypes="#//NamedElement">
  <eStructuralFeatures xsi:type="ecore:EReference" name="methods" upperBound="-1" eType="#//WebMethod"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="WebMethod" eSuperTypes="#//NamedElement">
  <eStructuralFeatures xsi:type="ecore:EReference" name="inputParameters" upperBound="-1" eType="#//Message"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="outputParameters" lowerBound="1" eType="#//Message"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="JavaEEWS">
  <eStructuralFeatures xsi:type="ecore:EReference" name="packages" upperBound="-1"/>
</eClassifiers>
</ecore:EPackage>
```
VI.4.2 JEE Model of Buyer/Seller Example

The Figure 6-6 is the JEE model of buyer/seller example.

Figure 6-6 JEE Model of Buyer/Seller Example
VI.4.3 ATL UPMS Model to JEE Model

Figure 6-7 UPMS Model of Buyer/Seller Example
VII Buyer-Seller Model Transformation Example (ATL)

ATL, the Atlas Transformation Language, is the ATLAS INRIA & LINA research group’s answer to the OMG MOF and QVT RFP. It is a model transformation language specified as both a meta-model and a textual concrete syntax. ATL provides developers with a mean to specify the way to produce a number of target models from a set of source models.

VII.1 Transformation Objectives

The objectives of the transformation using ATL is to transfer from model to model from Platform-independent model (PIM) level to Platform-specific model (PSM). In another word, we want to transfer model which with different hierarchy meta-model support.

VII.2 Transformation Technology

Until now, ATL and MOFScript have their specific plug-in under Eclipse platform.

An ATL transformation program is composed of rules that define how source model elements are matched and navigated to create and initialize the elements of the target models. Besides basic model transformations, ATL defines an additional model querying facility that enables to specify requests onto models. ATL also allows code factorization through the definition of ATL libraries.

VII.3 Transformation Theory

Totally, these transformations are all based meta-model run able. Utilizing transformation between meta-models and texts, we could construct the relation between different hierarchies such as PIM and PSM.

Model transformation aims to provide a mean to specify the way to produce target models from a number of source models. Formally, a simple model transformation has to define the way for generating a model Mb, conforming to a meta-model MMb, from a model Ma conforming to a meta-model MMa. As previously highlighted, a major feature in model engineering is to consider, as far as possible, all handled items as models. The model transformation itself therefore has to be defined as a model. This transformation model has to conform to a transformation meta-model that defines the model transformation semantics. As other meta-models, the transformation meta-model has, in turn, to conform to the considered meta-meta-model. (See Figure 7-1)

![Figure 7-1 Overview of model transformation](image-url)
VII.4 Transformation Rules

VII.4.1 Helper

ATL helpers can be viewed as the ATL equivalent to Java methods. They define factorized ATL code that can be called from different points of an ATL transformation. For example:

- helper context Integer def : double() : Integer = self * 2;
- helper context Families!Member def: familyName : String = …

VII.4.2 Rules

There are two type of rules: matched rules & called rules.

The matched rules constitute the core of an ATL declarative transformation since they make it possible to specify for which kinds of source elements target elements must be generated, and the way the generated target elements have to be initialized. For example:

```java
rule Author {
    from a : MMAuthor!Author
    to p : MMPerson!Person (
        name <- a.name,
        surname <- a.surname
    )
}
```

The called rules provide ATL developers with convenient imperative programming facilities. Called rules can be seen as a particular type of helpers: they have to be explicitly called to be executed and they can accept parameters. For example:

```java
rule NewPerson (na: String, s_na: String) {
    to p : MMPerson!Person (name <- na )
    do { p.surname <- s_na } }
```

VII.4.3 Main Method

ATL defines a number of collection data types that provide developers with different ways to handle collections of elements. The provided collection types are Set, OrderedSet, Bag and Sequence. Collection is the common abstract super-class of these different types of collections. For example: `OrderedSet(Tree!TreeElement) = ……`

The ATL specification also defines a number of iterative operations, also called iterative expressions, on the collection types. The main difference between a classical operation and an iterative expression on a collection is that the iterator accepts an expression as parameter, whereas operations only deal with data. For example:

- `Sequence{8, -1, 2, 2, -3}->iterate( e ; sum : Integer=0 | if…then…else…endif;
- `OrderedSet(MMTree!TreeElement) = self. children -> iterate (child ; elements: OrderedSet(MMTree!TreeElement) = OrderedSet{} | if…then…else elements.append(child); endif;

VII.5 ATL Example

Until now, we have implemented the transformation between UPMS light to JEE-WS. UPMS meta-model is the ‘MMa’, JEE-WS meta-model is the ‘MMb’, ‘Mt’ is the ATL file. ‘Ma’ is the source model which made based on ‘MMa’. We want to transfer Ma to Mb which based on Mata-model ‘MMb’ through ATL file. UPMS light is the Mata-model we built which contains the main elements in UPMS meta-model (See figure 7-2). Main elements in UPMS light meta-model are: Participant, Port, Interface, Operation, MessageType and Attribute.
We also built the JEE-WS meta-model which we could see as part of JEE architecture. (See figure 7-3). Main elements in JEE-WS meta-model are: Package, WebService, WebMethod, Message and Attribute.
With the source meta-model and target meta-model, we could build our ATL transformation now. Here we give the example we build, which defines the mapping rule between participant and package, Interface and web-service, operation and web-method, etc. Code is like this:

```plaintext
module UPMS2JEE;
create OUT : JEE from IN : UPMS;
rule Participant2Package {
  from upms : UPMS!Participant
to jee : JEE!Package (name <- upms.name,
  services <- upms.requisitions->collect (e | e.required),
  services <- upms.services->collect(e | e.provided))
}
rule Interface2WebService {
  from upms : UPMS!Interface
to jee : JEE!WebService (name <- upms.name,
  methods <- upms.operations)
}
rule Operation2WebMethod {
  from upms : UPMS!Operation
to jee : JEE!WebMethod (name <- upms.name,
  inputParameters <- upms.inParameters,
  outputParameters <- upms.outParameters)
}
rule MessageType2Message {
  from upms : UPMS!MessageType
to jee : JEE!Message (name <- upms.name,
  ownedAttributes <- upms.ownedAttributes)
}
rule Attribute2Attribute {
  from upms : UPMS!Attribute
to jee : JEE!Attribute (name <- upms.name)
```
After we got the source meta-model and target meta-model, we can build our ATL file and transfer model between PIM a PSM level. When we execute the ATL file, we have to configure the Source and target resource. (See figure 7-4)

![ATL Configuration](image)

The source model comes from the GMF editor, which basic on the UPMS meta-model. That means the elements in source model could all mapping with this meta-model. According to this transformation, it would generate a target model which based on JEE meta-model. This could be the input for MOFScript programming afterwards. Source model and target model are like these:

```xml
<?xml version="1.0" encoding="UTF-8"?>
    <ownedElements xsi:type="UPMS:Participant" name="Customer">
        <requisitions name="customerReprPort" required="/@ownedElements.6"/>
    </ownedElements>
    <ownedElements xsi:type="UPMS:Participant" name="Customer Manager">
        <services name="providedPort" provided="/@ownedElements.6"/>
        <requisitions name="reqPort" required="/@ownedElements.5 //@ownedElements.7"/>
    </ownedElements>
    <ownedElements xsi:type="UPMS:Participant" name="ProductManager">
        <services name="providedPort" provided="/@ownedElements.5"/>
    </ownedElements>
    <ownedElements xsi:type="UPMS:Participant" name="OrderManager">
        <services name="providedPort" provided="/@ownedElements.7"/>
        <requisitions name="requiredPort" required="/@ownedElements.9"/>
    </ownedElements>
</UPMS:ServiceDiagram>
```
Supplement on Buyer/Seller Purchase Order Example

```xml
<ownedElements xsi:type="UPMS:Participant" name="PaymantManager">
  <services name="providedPort" provided=" //@ownedElements.10"/>
</ownedElements>
<ownedElements xsi:type="UPMS:Interface" name="productService">
  <operations name="productList" outParameters=" //@ownedElements.14"/>
</ownedElements>
<ownedElements xsi:type="UPMS:Interface" name="customerRepresentingService">
  <operations name="authenticate" inParameters=" //@ownedElements.11"/>
  <operations name="recieveOrder" inParameters=" //@ownedElements.12" outParameters=" //@ownedElements.13"/>
</ownedElements>
<ownedElements xsi:type="UPMS:Interface" name="orderService">
  <operations name="processOrder" inParameters=" //@ownedElements.11" outParameters=" //@ownedElements.13"/>
</ownedElements>
<ownedElements xsi:type="UPMS:Participant" name="CRM">
  <services name="providedPort" required=" //@ownedElements.10"/>
  <requisitions name="requiredPort" provided=" //@ownedElements.9"/>
</ownedElements>
<ownedElements xsi:type="UPMS:Interface" name="crmService">
  <operations name="authenticateCustomer"/> 
  <operations name="processPayment" inParameters=" //@ownedElements.11 //@ownedElements.12"/>
</ownedElements>
<ownedElements xsi:type="UPMS:Interface" name="paymentService">
  <operations name="processPayment" inParameters=" //@ownedElements.11 //@ownedElements.12"/>
</ownedElements>
<ownedElements xsi:type="UPMS:MessageType" name="Customer">
  <ownedAttributes name="name"/>
  <ownedAttributes name="adress"/>
  <ownedAttributes name="creditcard"/>
</ownedElements>
<ownedElements xsi:type="UPMS:MessageType" name="Order">
  <ownedAttributes name="product"/>
</ownedElements>
<ownedElements xsi:type="UPMS:MessageType" name="Product"/>
<ownedElements xsi:type="UPMS:MessageType" name="ProductList"/>
</UPMS:ServiceDiagram>
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:javaeews="http://javaeews">
  <javaeews:Package name="Customer"/>
  <javaeews:Package name="Customer Manager">
    <services name="customerRepresentingService" methods="/7/8/9"/>
  </javaeews:Package>
  <javaeews:Package name="ProductManager">
    <services name="productService" methods="/6"/>
  </javaeews:Package>
  <javaeews:Package name="OrderManager">
    <services name="crmService" methods="/11/12"/>
    <services name="orderService" methods="/10"/>
  </javaeews:Package>
  <javaeews:Package name="PaymentManager">
    <services name="paymentService" methods="/13"/>
  </javaeews:Package>
  <javaeews:Package name="CRM"/>
  <javaeews:WebMethod name="productList" outputParameters="/17"/>
  <javaeews:WebMethod name="productList" outputParameters="/17"/>
  <javaeews:WebMethod name="authenticate" inputParameters="/14"/>
  <javaeews:WebMethod name="receiveOrder" inputParameters="/15" outputParameters="/16"/>
  <javaeews:WebMethod name="processOrder" inputParameters="/14" outputParameters="/16"/>
  <javaeews:WebMethod name="authenticateCustomer"/>
  <javaeews:WebMethod name="processPayment" inputParameters="/14/15"/>
  <javaeews:WebMethod name="processPayment" inputParameters="/14/15"/>
  <javaeews:Message name="Customer" ownedAttributes="/18/19/20"/>
  <javaeews:Message name="Order" ownedAttributes="/21"/>
  <javaeews:Message name="Product"/>
  <javaeews:Message name="ProductList"/>
  <javaeews:Attribute name="name"/>
  <javaeews:Attribute name="adress"/>
  <javaeews:Attribute name="creditcard"/>
  <javaeews:Attribute name="product"/>
</xmi:XMI>
VIII  Buyer-Seller Model Transformation Example (MOFScript)

The MOFScript subproject aims at developing tools and frameworks for supporting model to text transformations, e.g., to support generation of implementation code or documentation from models. It should provide a meta-model-agnostic framework that allows usage of any kind of meta-model and its instances for text generation.

VIII.1  Transformation Objectives

The objectives of the transformation using MOFScript is to transfer model to text from Platform-specific model (PSM) level to java code. In another word, we want to transfer model which build by JEE concept to direct text, such as java

VIII.2  Transformation Technology Background

As to MOFScript, it makes the language look like the most commonly used programming languages such as Java. A MOFScript transformation is written based on the source meta-model and is executed by the MOFScript tool on a source model at one meta-level lower. The code has been developed that supports parsing, checking, editing, and execution of MOFScript code. Currently, these are deployed as separate Eclipse plug-in components, which run within Eclipse. The parser and run-time may also be used independently of Eclipse.

VIII.3  Transformation Theory

The MOFScript tool is developed as two main logical architectural parts: tool components and service components (see Figure 8-1). The tool components are end user tools that provide the editing capabilities and interaction with the services. The services provide capabilities for parsing, checking, and executing the transformation language. The language is represented by a model (the MOFScript model), an Eclipse Modeling Framework (EMF) model populated by the parser. This model is the basis for semantic checking and execution. The MOFScript tool is implemented as an Eclipse plug-in using the EMF plug-in for handling of models and meta-models.

Figure 8-1 MOFScript architecture
VIII.4 MOFScript Rules

MOFScript transformation are groupings of transformation rules, they have one or more model parameters and have a name. One file may contain one or more text transformations and import other transformations.

VIII.4.1 Entry Point

There is always an entry point in MOFScript. For Example:

- `uml.Model::main () { // code for entry point }`
- `uml.Class::main () { // code for entry point }

The main() rule is implicitly invoked for each model element corresponding to the context type.

VIII.4.2 Conditional Loop

MOFScript supports three kinds of loop or condition: forEach, while and if-else. For example:

- `self.ownedMember->forEach(c:myMod.Class) {
  <$class name="" c.name ""/> 
}
- `while (myCounter > 0) {
  'counter value : ' counter 
}
- `if (self.hasStereotype("Feature") {
  ' This is a feature type '
else if (self.hasStereotype("Product")) {
  ' This is a product type '
else {
  ' this is neither ' 
}

Something deserves to mention, the forEach iterator is quite powerful as it applies to strings, integers, model types, collection types.

VIII.4.3 Collection

There are two kind of collection: List & Hashtable which is similar to Java List / Hashtable and can be used in forEach iterators. For example:

- `var packageNames_List:List
- var packageName_Hashtable:Hashtable
- self.ownedMember->forEach(p:uml.Package) {
  packageNames_List.add (p.name)
  packageName_Hashtable.put (p.id, p.name) }

The “->” symbol indicates that this is a collection. Typing “.” after a collection type will automatically insert the “->” symbol (e.g. ownedMember, Hashtable).

VIII.4.4 Text Output

There are two kinds of output: Prints and Escaped output. Prints are explicit print statements, they may be directed to stdout or a file. For example:

- `print ("output to current output file")`
- `stdout.println ("output to stdout")`
- `aFile.println ("output to file")`

Escaped output provides output in terms of list of string expressions. The string literals are combined with references and rules/functions. For example:

- `public class c.name extends Serializable {`

VIII.5 Buyer-Seller Example

The source we use comes from ATL transformation result. Such as meta-model and source model. Meta-model is the JEE meta-model and source model is the transformation result of ATL. Here we present part of the MOFScript code:

```wtext
transformation Transformation (in mdl:"classModel") {
  mdl.Model::main () {
    self.ownedPackages->forEach(package : mdl.Package){
      package.mapPackage()
    }
  }
  mdl.Package::mapPackage()
```

After finishing compiling the transformation, any errors in the transformation will be presented. Fix errors and execute the MOFScript file (such as figure 8-2).
After executing the MOFScript file, we could generate the java file. Figure 8-3 is the generated package, inside the package, there are java files which maps the JEE elements in Platform-specific Model (PSM).

Here is the java file example for OrderManager, which is a package element in JEE platform.

```java
package OrderManager;
import java.util.ArrayList;

public class OrderService {
    public Product processOrder(Customer customer0) {
        return null;
    }
}
```