A UML Profile for Enterprise Distributed Object Computing

Joint Final Submission

Part II Supporting Annexes

Version 1.0

Revised 22 August 2001

Submitted by:

CBOP
Data Access Technologies
DSTC
EDS
Fujitsu
IBM
Iona Technologies
Open-IT
Sun Microsystems
Unisys

Supported by:

Hitachi
SINTEF
NetAccount

OMG Document Number: ad/2001-08-20

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by sending email to issues@omg.org. Please reference precise page and section numbers, and state the specification name, version number, and revision date as they appear on the front page, along with a brief description of the problem. You will not receive any reply, but your report will be referred to the OMG Revision Task Force responsible for the maintenance of the specification. If you wish to be consulted or informed during the resolution of the submitted issue, indicate this in your email. Please note that issues appear eventually in the issues database, which is publicly accessible.

Contents

Part II – Supporting Material

1. Introduction

This part of the Joint UML for EDOC submission contains the following non-normative Annexes:

- Annex A - Procurement, Buyer/Seller example
- Annex B - Meeting Room example
- Annex C - Hospital example
- Annex D - Examples of Patterns
- Annex E - Technology mappings from EDOC to Distributed Component and Message Flow Platform Specific Models

In addition, XMI and DTD data files for the metamodels in the EJB/Java/FCM profiles are included in the zip file containing this Part of the submission, in the folder named “XMI and DTDs”.

Annex A – Procurement Process and Buyer/ Seller Example

Contents

List of Figures

1. Introduction

This Annex contains two linked examples. The first, in Section 2, is a specification of a system for describing and supporting the processes for procuring goods or services, modeled using the Business Processes profile (Part I, Chapter 3, Section 5). This calls up the second example, which uses the CCA Profile (Part I, Chapter 3, Section 2) to model in detail the BuySell process.

2. The Procurement System Example

This section contains a specification of a system for describing and supporting the processes for procuring goods or services for an organization. An informal textual description of the system is given followed by specifications of the business processes, business entities, rules and events involved in this system.

This example has been developed in collaboration with Mincom Limited and represents the expression of the business processes used by the company for sourcing goods. We thank Mincom for their assistance.

2.1 An Informal Description

The procurement system is concerned with the procurement of goods or services by an organization. The process for acquiring some resource (or service) can be started in either of two ways. In both cases, a request listing the resource requirements is received. In one case this is sufficient, however in the second case, the request is accompanied by additional information about the preferred freight options for delivery.

In both cases, the resource requirements are used as a basis for sourcing a number of potential suppliers of the goods. This list of potential suppliers (and for the second case, a corresponding list of freight sources) is then evaluated. The evaluation is a sophisticated process involving ranking and checking of potential suppliers. As a result of the evaluation, a supplier is awarded the contract to supply the required goods. Both the sourcing of potential suppliers and the evaluation process are the responsibility of the Purchasing Officer.

After the Authorizing Officer has awarded the contract to a particular supplier, the order is released to that supplier for processing. While the order is being processed, it is monitored to ensure that progress is made and the contract is fulfilled. Finally, after the resources are received, the receipt of the goods is approved, and any claims for payment are fulfilled.

2.2 The Business Process Model

The Procurement Business Process as shown in Figure 1 provides for Resource Requirements to be satisfied from various sources for a given request.

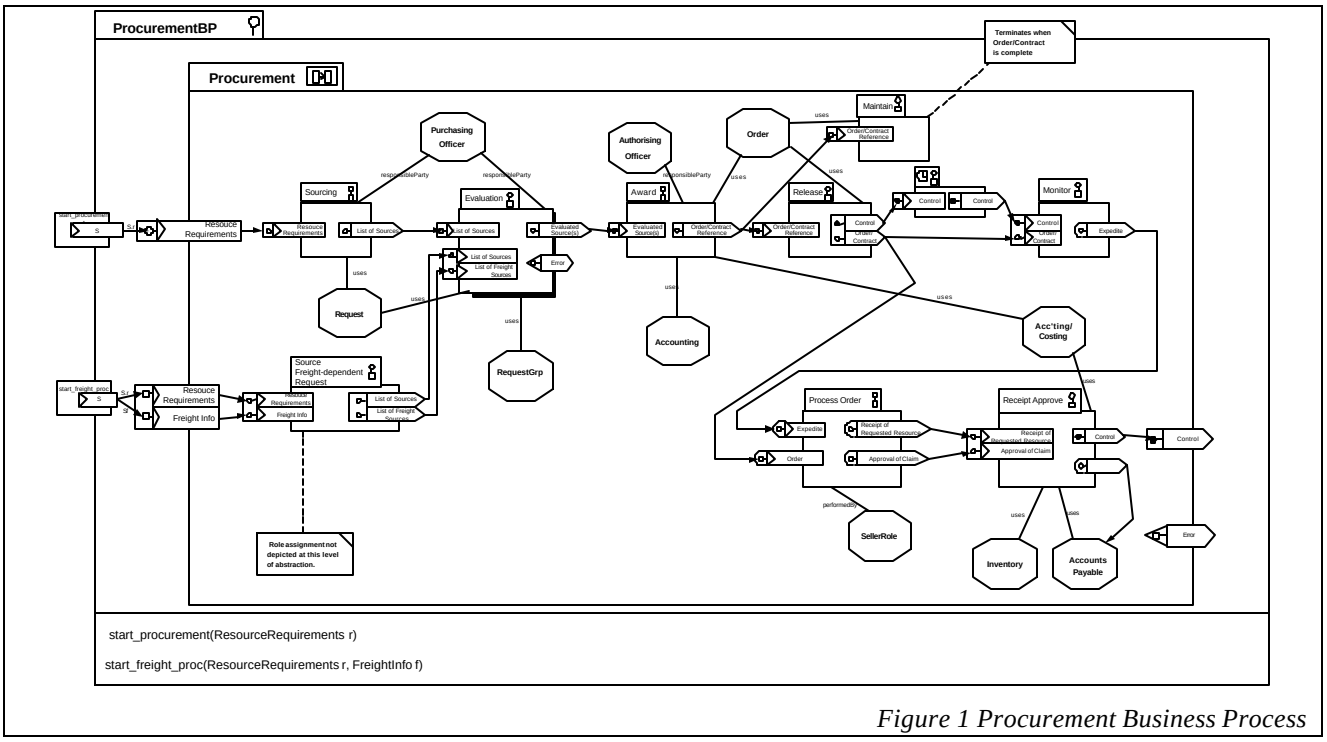


Figure 1 Procurement Business Process

The Procurement Business Process can be initiated in one of two ways by the invocation of one of its two operations. The input parameters are then used to enable one of the two alternative input sets on the Activity (and its in-line CompoundTask definition) specifying:

- Resource Requirements, or
- Resource Requirements plus Freight Requirements information if the request includes freight requirements.

The process completes successfully once sources for satisfying the Resource and Freight Requirements have been identified and evaluated, a contract has been awarded, released and processed, and finally the goods have been received and paid for.

Where no valid source can be found to satisfy the resource or freight requirements, the process will throw an appropriate user-exception indicating this and the process will terminate unsuccessfully.

The Procurement Business Process is modeled as being comprised of a number of Activities and CompoundTask definitions. These are discussed in detail in the following sections.

2.3 Detailed Task Description

Unless otherwise mentioned, all the following sections will refer to Figure 1.

2.3.1 Sourcing and Sourcing Freight-Dependent Request Processes

Both the Sourcing and the Sourcing Freight-Dependent Request processes fulfill the task of determining a list of potential sources for satisfying the Resource Request. Both processes will reference sourcing policies applicable to the request as well as referencing the Request itself. The association to the Request ProcessRole is shown as a *usesArtifact* relation - that is, the request is referenced as an artifact role. This relation is annotated with an 'R' to indicate that the access is a read-only operation.

The only distinction between the two tasks is that the Sourcing Freight-Dependent Request process has the additional work of considering the freight-requirements specified in the additional input to the task. Correspondingly it produces a list of sources for freight in addition to the list of sources for satisfying the resource request.

2.3.2 Evaluation

Having identified appropriate potential sources of supply, an evaluation is performed in accordance with the sourcing policy. This evaluation process completes successfully with output of the recommended preferred sources in a list ordered by the priority ranking of each source. The task can terminate with an exception when no valid sources can be found.

The CompoundTask definition for the Evaluation Activity has been elided from the Procurement process model for clarity of expression. It presents more fine-grained detail than the rest of the Procurement processes and rather than show this extra detail in the Procurement process, it is removed to the separate diagram Figure 2.

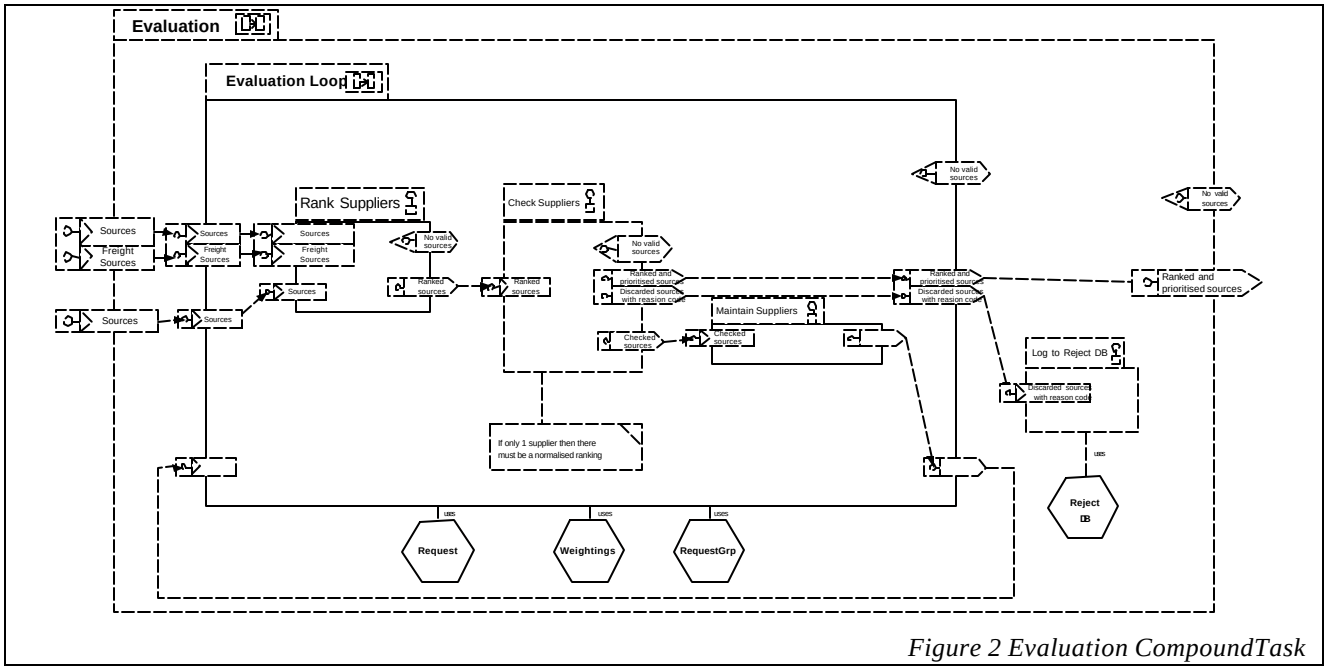


Figure 2 Evaluation CompoundTask

The Evaluation process is modeled as comprising an Evaluation Loop that iterates over a list of potential sources until either a prioritized list is produced, or the process is unable to find any valid sources and terminates with an exception.

The Evaluation Loop has three InputGroups. Two are for inputting the list of sources, and the list of sources accompanied by the list of freight sources. The third InputGroup has an Input that is a list of maintained and evaluated sources that will be subject to further evaluation.

The Evaluation Loop has two OutputGroups. The first has two output ProcessFlowPorts - a list of ranked sources and a list of discarded sources. The list of discarded sources is passed to an input ProcessFlowPort of the Log to Reject DB process that records details regarding the rejection of sources. The second OutputGroup of the Evaluation Loop has a single ProcessFlowPort that is a list of maintained or altered and evaluated sources that will be subject to further evaluation. This ProcessFlowPort is connected by a DataFlow back to one of the InputGroups allowing for re-iteration over the list of sources.

The Evaluation Loop makes use of a number of Artifact roles. It uses the Request, Weightings of the sources, and possibly the Request Group that is related to a specific request to assist in the evaluation. The Evaluation Loop terminates when all of the potential Sources have either been ranked and prioritized, or added to the list of discarded sources.

2.3.3 Award

The Award process takes as input the list of Evaluated Sources. From this list, the selected supplier is assigned to the Request and an order, contract or contract release is created as an Artifact ProcessRole. The Activity produces as output a reference to the Artifact role representing either the order, the contract, or the contract release.

The Award Activity is performed by the Authorizing Officer ProcessRole.

Commitment details about the order, contract or contract release are passed to the Accounting artifact ProcessRole.

Both the Maintain and the Release Activities may start concurrently after the Award Activity has enabled its output as they are both connected by DataFlows from the output ProcessFlowPort of this Activity.

2.3.4 Maintain

The Maintain Activity supports the maintenance of the Orders, Contracts or Contract Releases. It takes as input an identifier for an Order, Contract or Contract Release and uses this reference to read and possibly modify the actual data. The Maintain process uses the identified Order, Contract or Contract Release as an Artifact ProcessRole. Basically this process exists in recognition that Order, Contract or Contract Release are not completely static or stable and will need modification due to unforeseen circumstances.

This process has no output ProcessFlowPorts.

2.3.5 Release

The Order/Contract or Contract Release is forwarded to the selected supplier as part of the Release Activity. The Activity takes an identifier for an Order, Contract or Contract Release as input and passes this identifier on as output.

The termination of the Release Activity enables a Timer Task to start when it receives a signal via a Connection from the OutputGroup of the Release Activity. The Timer Task is used to enforce a delay on the enabling of the Monitor Activity.

2.3.6 Monitor

The Monitor Activity provides mechanisms to monitor supplier performance for timely delivery of the goods or services. It also monitors compliance with the terms of the Order, Contract or Contract Release.

The process has a single Asynchronous OutputGroup that will produce some notification to the supplier to expedite delivery.

2.3.7 Process Order

The BuySellCp ProcessRole performs the Process Order Activity. The Process Order Activity represents the actual supply of the goods or services to satis fy the order.

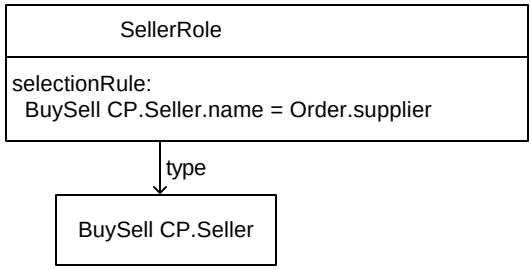


Figure 3 The SellerRole Performer Role

The BuySell CommunityProcess is specified in Section 3 of this Annex. The three Roles in the Community Process are played by the identity of the invoker of the Procurement BusinessProcess (self), and by the selected Supplier and freightSupplier. The only other input required to initiate the BuySell protocol between these Roles is the Order itself. Not shown here or in Figure 1 are the *represents* associations between the PortConnectors and the ProtocolPorts of the BuySell CommunityProcess. For example, the PortConnector of the Process Order Activity, labeled Order, to the ProtocolPort of the SalesProtocol, labeled OrderBT.

Asynchronously, the Supplier will supply goods to satisfy the order and will also generate notification of invoices that require payment for the delivery of the goods.

2.3.8 Receipt and Approve

The Receipt and Approve Activity handles the receiving goods, updating the inventory to reflect this, and the payment of invoices for the goods.

This process has an InputGroup comprising of a details relating to the receipt of the requested resource, and a request for approval of a claim for payments from the supplier.

3. The Sales example

This example illustrates the specification of a system of collaborating parties, involved in a commercial Sale.

The Sales example defines the collaboration between the parties involved.

The focus is on the boundaries between the parties – ComponentUsages, their specification – ProcessComponents, their connectable point – Ports, and the externally observable contract of candidate interactions – Protocols .

Each party may be further specified as an internal composition of collaborating sub-components, onto which the external contract is delegated.

3.1 Performer for the ProcessOrder Activity of the Procurement System example

The Sales example is referenced as part of the Procurement Process of the Buyer, as the Performer for the ProcessOrder Activity..

Please refer to the Procurement System example of the Business Processes Profile (Section 2 above), for the specification of the Business Process of the Buyer, where this Sales example is used and initiated, to fulfill the ProcessOrder Activity.

In the context of the Buyer Business Process :

(copied from the Procurement System example (Section 2))

"… After the Authorizing Officer has awarded the contract to a particular supplier, the order is released to that supplier for processing. …"

The organization performing the Procurement Process plays the role of Buyer, and the awarded supplier plays the role of Seller, in the BuySellCommunity CommunityProcess.

The Award Activity will determine the identity of the actual Seller instance, corresponding to a ProcessComponent type of Seller, that plays the Seller role in the BuySell CommunityProcess.

3.2 BuySell Community Process

The BuySell CommunityProcess specifies how a Buyer, a Seller and a Logistics collaborate to complete a business. Each role is played by a ComponentUsage of the same name. The specifications for the used ProcessComponent can be found under headers below.

The Buyer collaborates directly with the Seller, through the Buy and Sell ProtocolPorts, according to the Sales Protocol.

The Seller and the Buyer collaborate with the Logistics, through the Ship and Delivery ProtocolPorts, according to Protocol of the same names. The specification for the Protocols can be found under headers below.

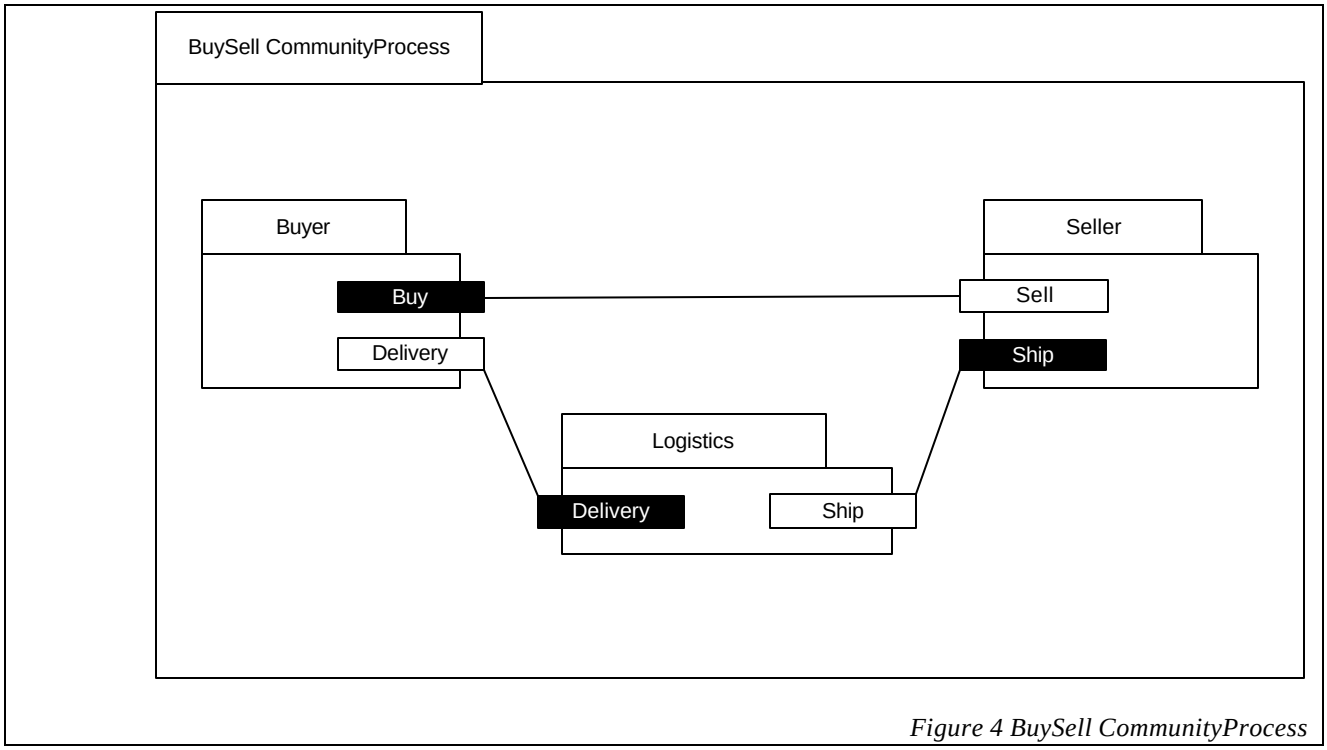


Figure 4 BuySell CommunityProcess

The activities in the BuySell Community Process start by the Buyer initiating the interactions on its Buy ProtocolPort, according to the Sales Protocol.

The Seller is connected through its Sell ProtocolPort, to the Buy ProtocolPort of the Buyer. Therefore, the Seller will respond to the Sales Protocol, as initiated from the Buyer.

The Seller will follow the Sales Protocol, and eventually initiate the Ship Protocol with the Logistics role. The Logistics role will respond to the Ship Protocol, and initiate the Delivery Protocol on the Buyer. The Buyer will then be able to proceed with the Sales Protocol, and complete the overall collaboration.

3.3 Protocols

3.3.1 Sales Protocol

The interactions between the ComponentUsage in the BuySell CommunityProcess, above, occur according to Protocols, as specified below.

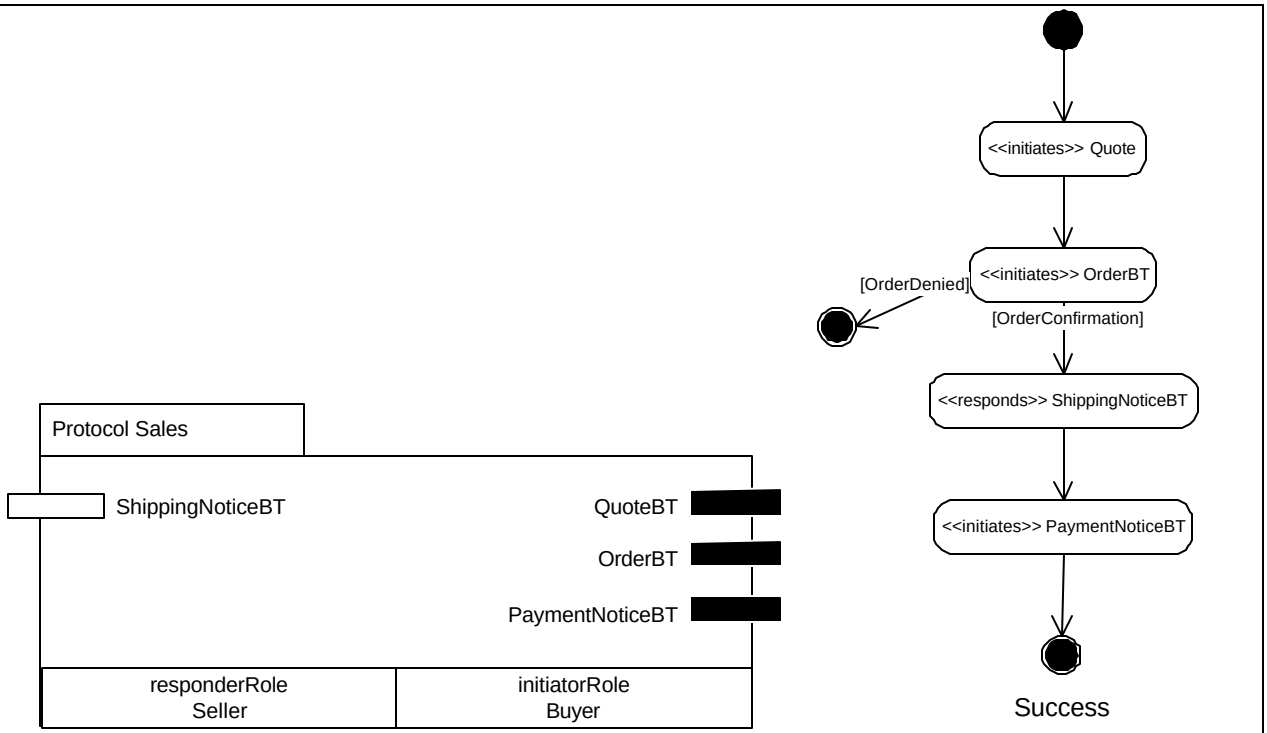


Figure 5 Sales Protocol structure and choreography

Structure

The Sales Protocol is an integration of four simpler Protocols : QuoteBT, OrderBT and PaymentNoticeBT. The Sales Protocol has a ProtocolPort using each of these simpler Protocols. The specification for these Protocols can be found under headers below.

Interactions in the ProtocolPorts QuoteBT, OrderBT and PaymentNoticeBT will be initiated by the initiatorRole of the Sales Protocol.

The initiatorRole of the Sales Protocol will respond to interactions in the ShippingNoticeBT ProtocolPort.

Choreography

Interactions in the Sales Protocol will begin by the initiatorRole of the Sales Protocol, initiating and fully performing the interactions of the QuoteBT ProtocolPort.

After this, the initiatorRole will initiate and fully perform the interactions of the OrderBT ProtocolPort.

If during performance of the interaction of the OrderBT ProtocolPort, an OrderDenied has flown between initiatorRole and responderRole, then the Protocol ends with a Failure condition.

Else, if an OrderConfirmation has flown, then the initiatorRole of the Sales Protocol will respond and fully perform the interactions of the ShippingNoticeBT ProtocolPort.

After this, the initiatorRole will initiate and fully perform the interactions in the PaymentNoticeBT ProtocolPort.

3.3.2 QuoteBT Protocol

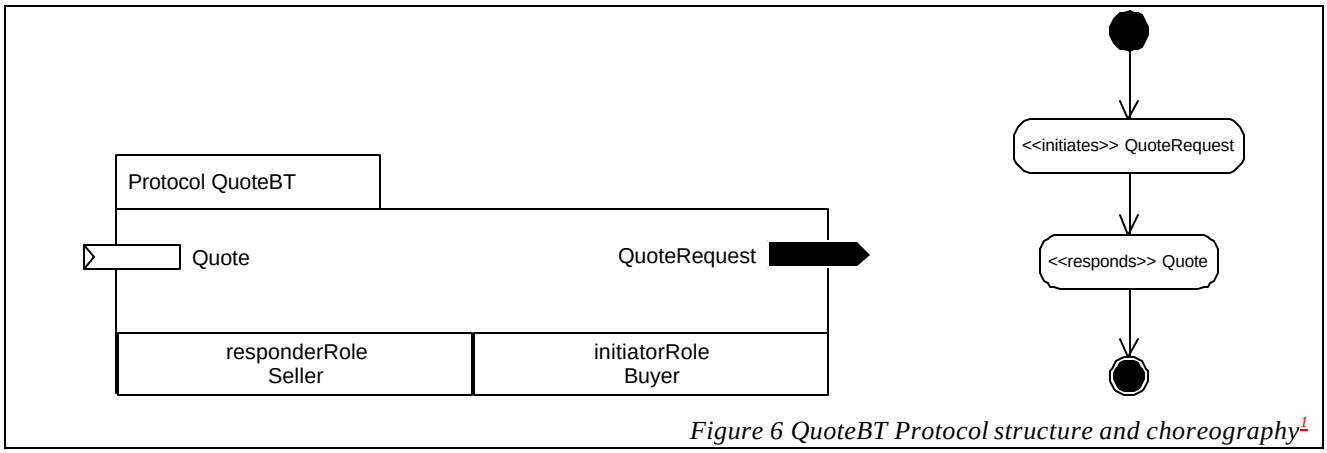


Figure 6 QuoteBT Protocol structure and choreography[1]

QuoteBT is a Protocol in the form of a Request-Reply, where the initiatorRole will send a QuoteRequest, and receive a Quote as response. QuoteRequest and Quote are FlowPort of the QuoteBT Protocol, typed to CompositeData of the same name.

3.3.3 OrderBT Protocol

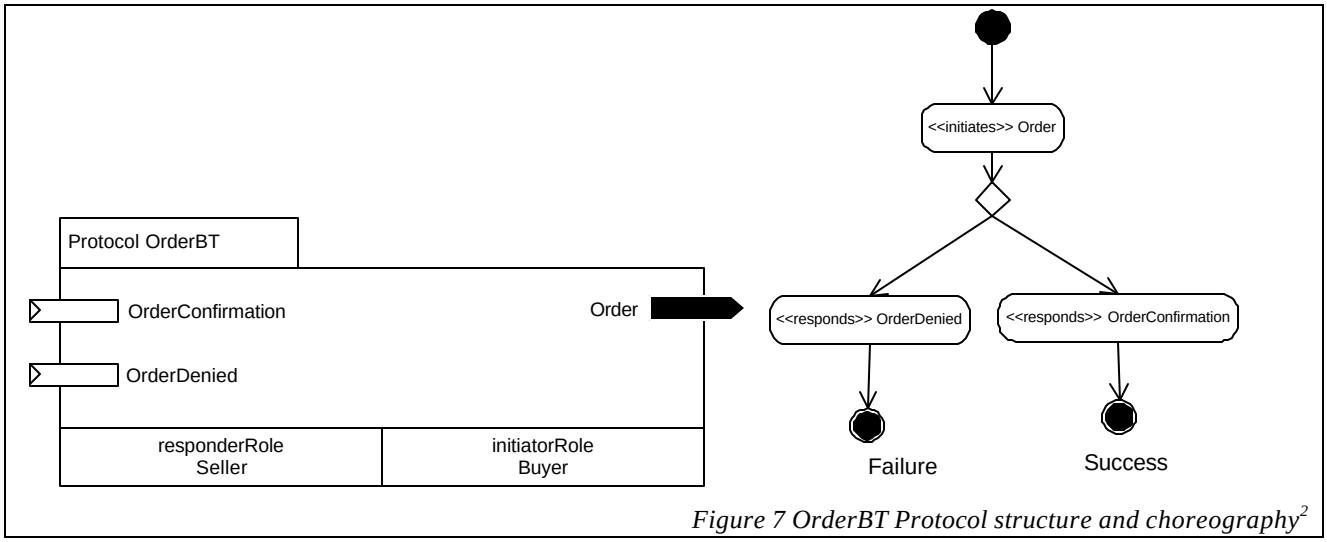


Figure 7 OrderBT Protocol structure and choreography[2]

QuoteBT is a Protocol in the form of a Request-Multiple_Candidate_Reply, where the initiatorRole will send an Order, and receive as response an OrderConfirmation or an OrderDenied. Order, OrderConfirmation and OrderDenied are FlowPort of the OrderBT Protocol, typed to CompositeData of the same name.

[1] The direction of the ports is incorrect in Figures 6 to 11. In all these diagrams, <<responds>> should read <<initiates>>, and *vice versa*.

[2] See footnote to Figure 6

An OrderConfirmation leads to a successful termination of the Protocol, while an OrderDenied is a Failure condition.

3.3.4 ShippingNoticeBT Protocol

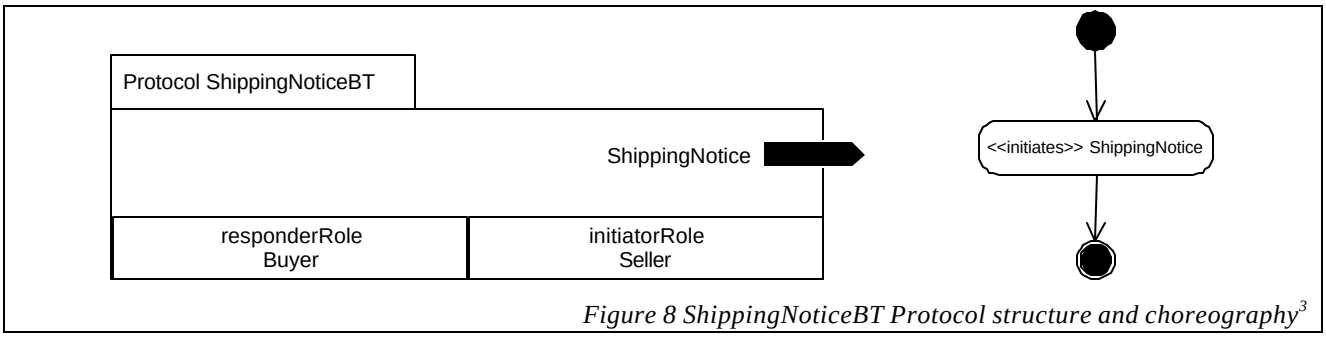


Figure 8 ShippingNoticeBT Protocol structure and choreography[3]

ShippingNoticeBT is a Protocol with a single FlowPort, corresponding to the sending of a ShippingNotice by the initiatorRole of the Protocol.

To declare a Protocol for a single flow may be redundant, as the unique FlowPort could be included wherever the Protocol is used, like in the Sales Protocol of our example. In this case, ShippingNoticeBT has been defined, for symmetry, and to illustrate the benefit of this approach, encapsulating as a Protocol the single flow nature of the interaction.

3.3.5 PaymentNoticeBT Protocol

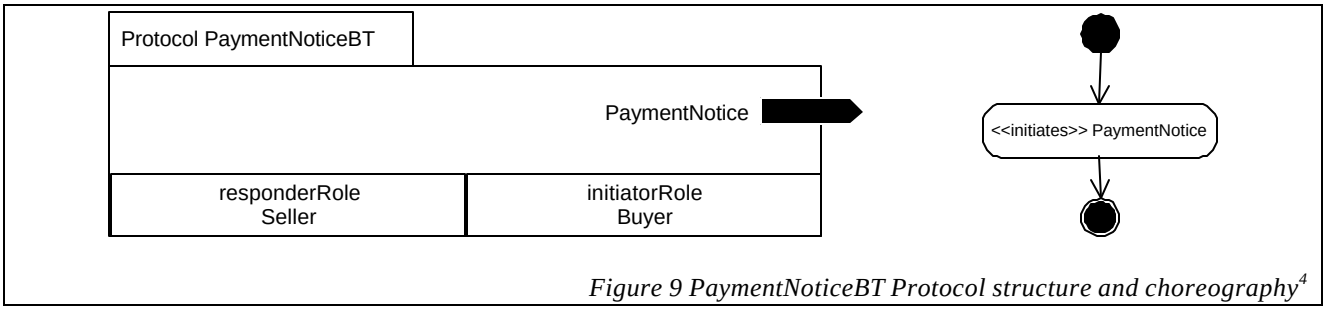


Figure 9 PaymentNoticeBT Protocol structure and choreography[4]

PaymentNoticeBT is a Protocol with a single FlowPort, corresponding to the sending of a PaymentNotice by the initiatorRole of the Protocol.

[3] See footnote to Figure 6
[4] See footnote to Figure 6

3.3.6 ShipBT Protocol

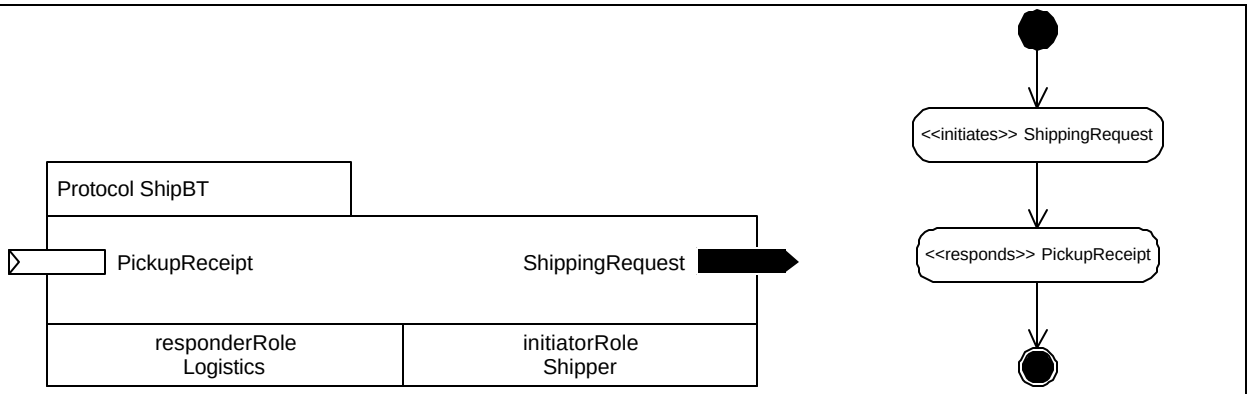


Figure 10 ShipBT Protoco structure and choreography[5]

ShipBT is a Protocol in the form of a Request-Reply, where the initiatorRole will send a ShippingRequest, and receive a PickupReceipt as response. ShippingRequest and PickupReceipt are FlowPort of the ShipBT Protocol, typed to CompositeData of the same name.

3.3.7 DeliveryBT Protocol

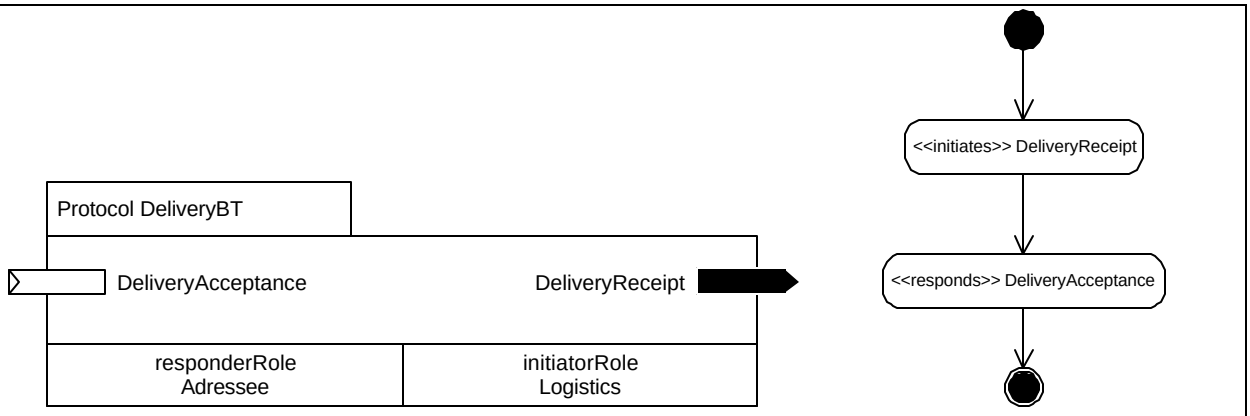


Figure 11 DeliveryBT Protocol structure and choreography[6]

DeliveryBT is a Protocol in the form of a Request-Reply, where the initiatorRole will send a DeliveryReceipt, and receive a DeliveryAcceptance as response. DeliveryReceipt and DeliveryAcceptance are FlowPort of the DeliveryBT Protocol, typed to CompositeData of the same name.

[5] See footnote to Figure 6
[6] See footnote to Figure 6

3.4 Components

3.4.1 Buyer ProcessComponent

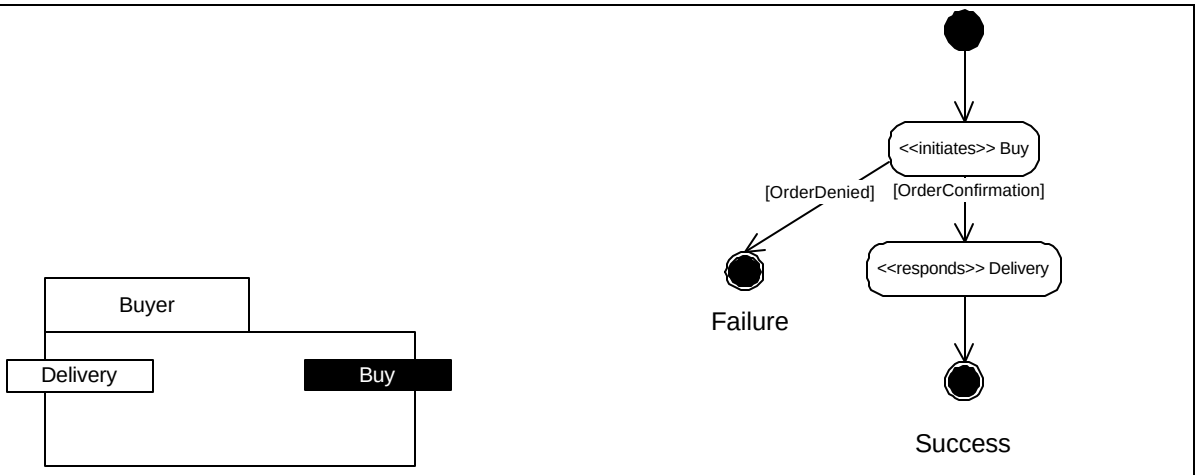


Figure 12 Buyer ProcessComponent structure and choreography

Buyer ProcessComponent is used in the BuySell CommunityProcess, as ComponentUsage of the same name.

Buyer has two ProtocolPort named Buy and Delivery.

The Buyer initiates interactions through the Buy ProtocolPort according to the Sales Protocol. The Delivery ProtocolPort responds to the DeliveryBT Protocol.

The activities of the Buyer ProcessComponent will begin by initiating and fully performing the interactions through the Buy Port, according to the used Sales Protocol.

After this, if during performance of the interaction of the Sales Protocol through the Buy ProtocolPort, an OrderDenied has flown, then the choreography ends with a Failure condition.

Else, if an OrderConfirmation has flown, then the Buyer ProcessComponent will respond to interactions through the Delivery ProtocolPort, and complete successfully.

3.4.2 Seller ProcessComponent

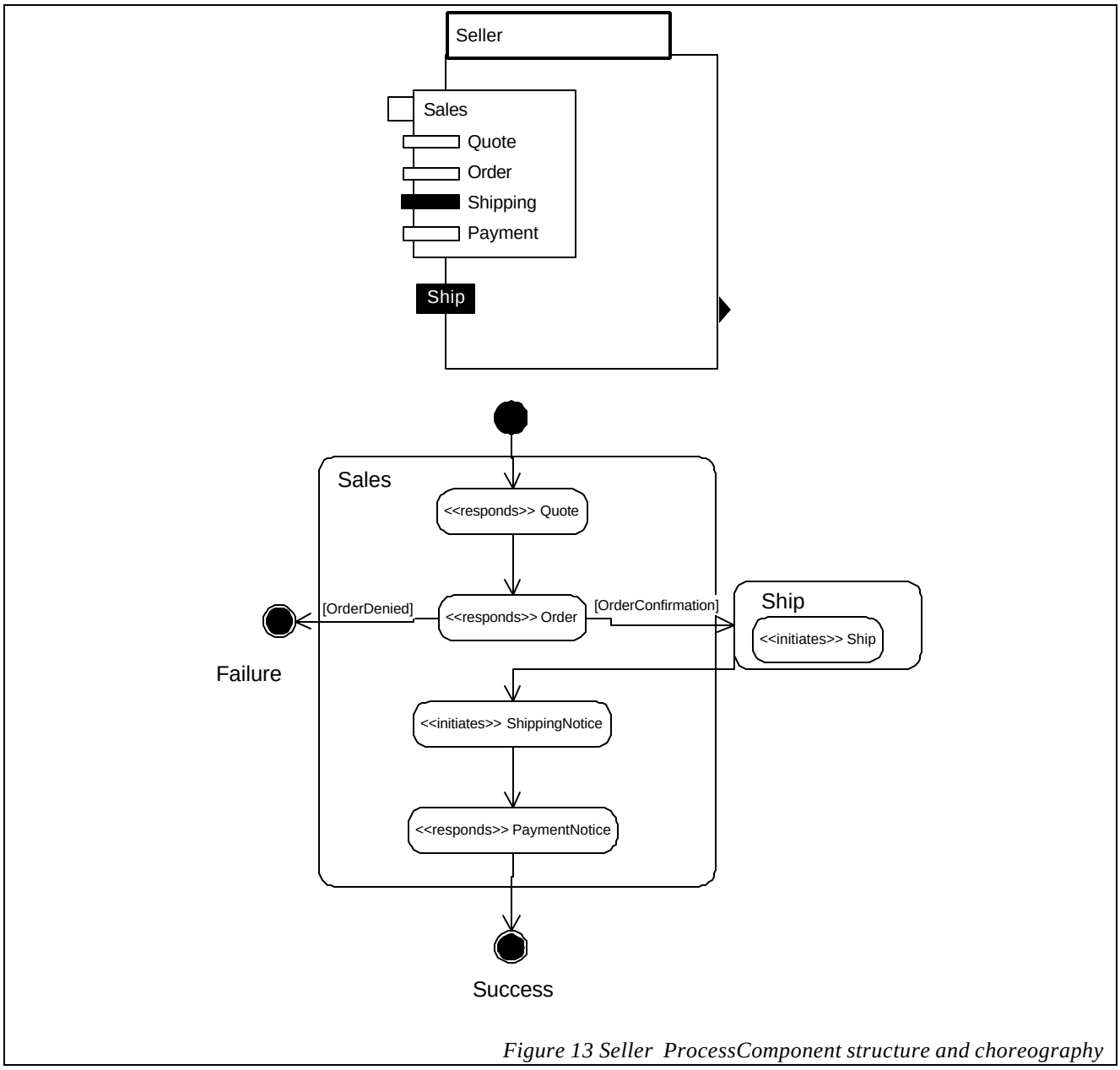


Figure 13 Seller ProcessComponent structure and choreography

Seller ProcessComponent is used in the BuySell CommunityProcess, as ComponentUsage of the same name.

Seller has two ProtocolPort named Sell and Ship.

The Seller responds to interactions through the Sell ProtocolPort according to the Sales Protocol. The Ship ProtocolPort initiates interactions in the Delivery Protocol.

The activity of the Seller ProcessComponent will begin when responding and fully performing the interactions through the Buy Port, according to the used Sales Protocol.

The Failure termination condition of the Sales Protocol is also a Failure termination condition of the choreography of the Seller ProcessComponent.

In the choreography for the Seller ProcessComponent, the interactions through the Ship ProtocolPort, according to the ShipBT Protocol, are inserted as a whole in between two consecutive states of the Sales Protocol in the Sell ProtocolPort.

The choreography of the Seller ProcessComponent is an integration of the choreographies of the Sales and ShipBT Protocols, of the Sell and Ship ProtocolPort. The integration is safely achieved by insertion, as a refinement of a Transition in the Sales Protocol, as two Transitions to and from the inserted Ship PortActivity.

The interactions through the Sell ProtocolPort are integrated with the Ship ProtocolPort, by insertion of the whole ShipBT Protocol, interleaved between two activities of the Sales Protocol. This is a case of safe synthesis, where the constraints and partial ordering of each Protocol are still valid in the synthesized protocol.

The successful termination of the choreography of the Sales Protocol in the Sell ProtocolPort, is also the successful termination of the Seller ProcessComponent.

This structure and choreography fully specify the external contractual obligations and expectations of the Seller ProcessComponent.

No details have been offered, about how the Seller ProcessComponent actually performs its duties, in compliance with the externally observable structure and behavior specified above.

3.4.3 Seller ProcessComponent – internal composition

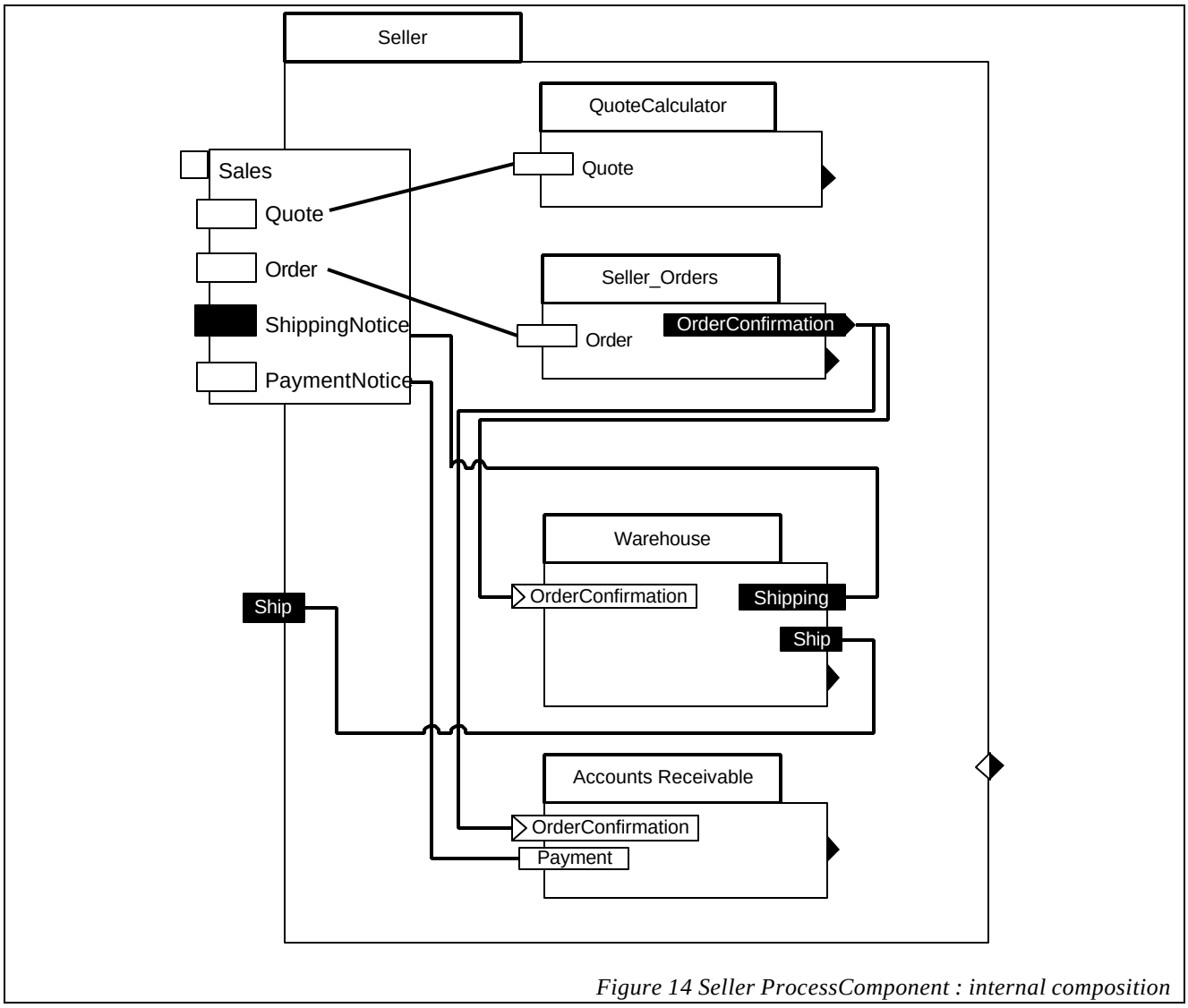


Figure 14 Seller ProcessComponent : internal composition

In the header above, the externally observable structure and choreography have been defined, without revealing any internal details of the Seller ProcessComponent.

When designing a system, that will play the Seller role in a BuySell CommunityProcess, the Seller ProcessComponent will have to be further specified, and its complexity decomposed in smaller units – and recursively – until the resulting ProcessComponent can be directly mapped or implemented to non-CCA artifacts.

The internal de-composition of the Seller ProcessComponent, must comply with the externally observable choreography. If it complies, the Seller may play the role in the BuySell Community Process – and others using the Seller ProcessComponent definition – independently of how the Seller ProcessComponent has been internally defined.

In our example, the Seller ProcessComponent is internally composed by using QuoteCalculator, Seller_Order, Warehouse and AccountsReceivablel components.

The Sell ProtocolPort is rendered expanded, displaying the ProtocolPort of the Sales Protocol, as sub-Port of the Sell ProtocolPort.

The individual sub-ProtocolPort of Sell are delegated or initiated to/from port of sub-component of Seller.

The usage of QuoteCalculator responds to and handles the Quote sub-port of Sell. The QuoteCalculator ProcessComponent has a ProtocolPort using the QuoteBT Protocol, and is therefore compatible for direct delegation from the Quote sub-port of Sell.

Similarly, the Seller_Orders component usage responds to and handles the Order sub-Port of Sell. In addition, the Seller_Orders ProcessComponent has an additional OrderConfirmation outgoing flow, connected to the Warehouse and AccountsReceivable component usages. When Seller_Orders responds an OrderConfirmation, the same OrderConfirmation will be sent to Warehouse and AccountsReceivable.

The Warehouse component usage responds to the OrderConfirmation from the Seller_Orders component, and initiates the interactions of the ShipBT Protocol, forwarded through the Ship ProtocolPort of the container Seller ProcessComponent. After, the Warehouse component initiates the interactions of the ShippingNoticeBT Protocol, through the ShippingNotice sub-Port of Sell.

The AccountsReceivable component usage receives OrderConfirmation from Seller_Orders, and responds to and handles the PaymentNotice sub-port of Sell.

3.4.4 QuoteCalculator ProcessComponent

The QuoteCalculator ProcessComponent has the structure as shown in its component usage in the Seller internal compositions.

QuoteCalculator has a single ProtocolPort responding to the QuoteBT Protocol.

The chorography of QuoteCalculator corresponds to the choreography of the QuoteBT Protocol.

3.4.5 Seller_Orders ProcessComponent

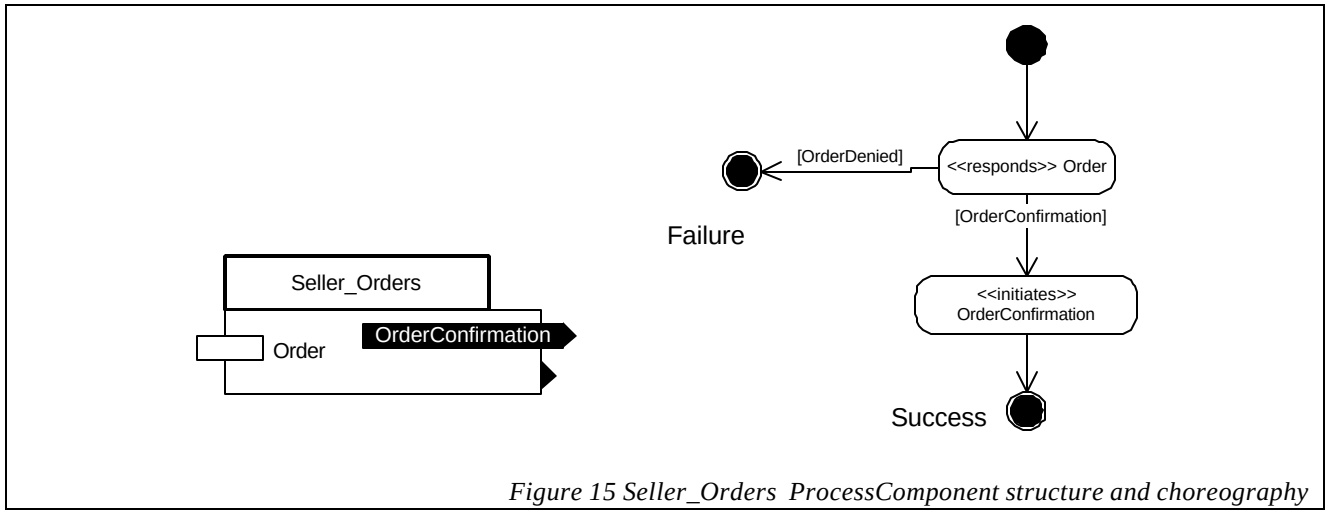


Figure 15 Seller_Orders ProcessComponent structure and choreography

Seller_Orders ProcessComponent responds to interactions of the OrderBT Protocol through the Order ProtocolPort.

The Seller_Orders ProcessComponent has an additional OrderConfirmation outgoing flow. When Seller_Orders responds an OrderConfirmation, the same OrderConfirmation will be sent also through the FlowPort.

3.4.6 Warehouse ProcessComponent

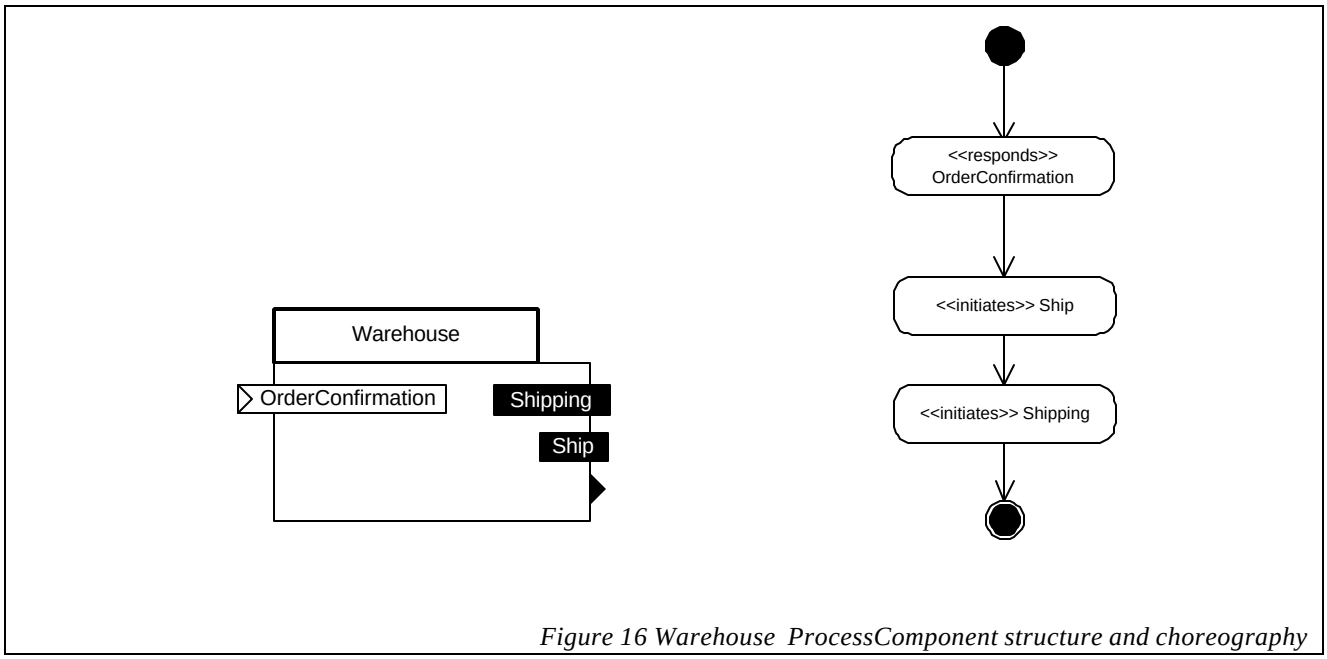


Figure 16 Warehouse ProcessComponent structure and choreography

The Warehouse ProcessComponent receives an OrderConfirmation flow, and initiates the interactions of the ShipBT Protocol, through the Ship ProtocolPort. After, the Warehouse component initiates the interactions of the ShippingNoticeBT Protocol, through the ShippingNotice Port.

3.4.7 AccountsReceivable ProcessComponent

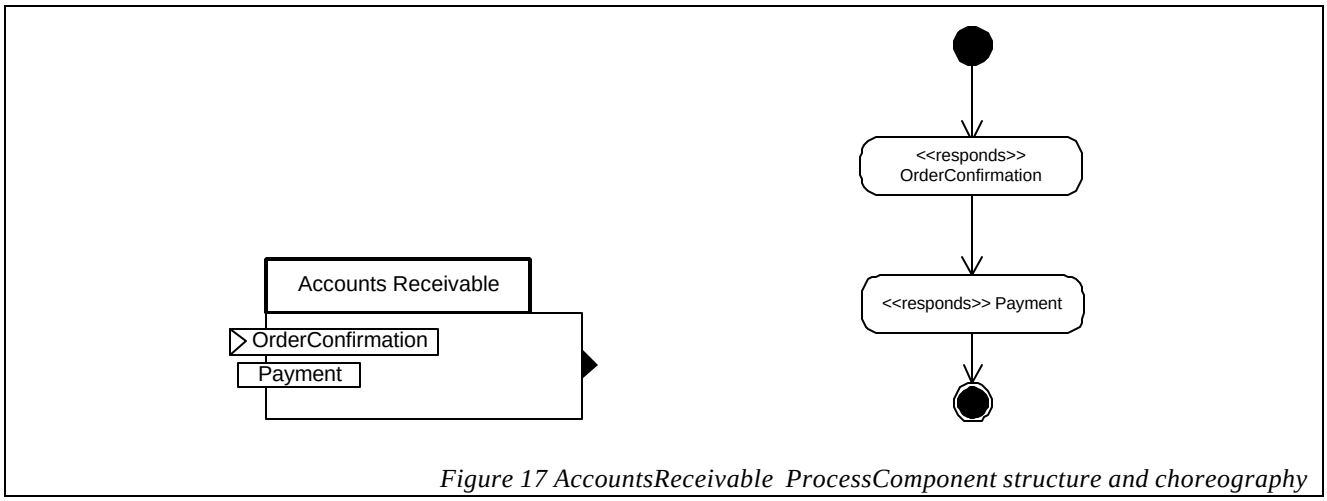


Figure 17 AccountsReceivable ProcessComponent structure and choreography

The AccountsReceivable ProcessComponent receives an OrderConfirmation, and responds to the PaymentNoticeBT Protocol through the Payment ProtocolPort.

3.4.8 Logistics ProcessComponent

Figure 18 Logistics ProcessComponent structure and choreography

Logistics ProcessComponent is used in the BuySell CommunityProcess, as ComponentUsage of the same name.

Logistics has two ProtocolPort named Ship and Delivery.

The Logistics responds to interactions through the Ship ProtocolPort according to the ShipBT Protocol. The Delivery ProtocolPort initiates interactions of the DeliveryBT Protocol.

The activities of the Logistics ProcessComponent will begin by responding and fully performing the interactions through the Ship Port, according to the used ShipBT Protocol.

After this the Logistics ProcessComponent will initiate and fully perform the interactions through the Delivery ProtocolPort.

The Logistics ProcessComponent integrates the ShipBT and DeliveryBT Protocols, by safely synthesizing them in a sequence, where the ShipBT Protocol is fully exercised and completed, before starting the DeliveryBT Protocol.

Annex B – The Meeting Room Example

Contents

List of Figures

3.5 Introduction

This annex describes and specifies a Meeting Reservation System (MRS) in terms of the UML Profile for EDOC. The ISO RM-ODP framework (The Enterprise, Information, Computational, Engineering and Technology Viewpoint) is used to structure the MRS specification.

The model for the Meeting Reservation System comes from the COMBINE (COMponent-Based Interoperable Enterprise system development) project, where it function as the small-grained pilot for proving the COMBINE concepts.

The overall goal of COMBINE (ESPRIT project no. IST-1999-20893) is to support model-driven development of enterprise systems - using components. This requires further development of methods, infrastructures and tools as well as business solutions for modeling, designing, deploying, testing and running components successfully in an enterprise-wide scale. The UML profile for EDOC will form a baseline for the COMBINE project.

3.5.1 Description

The Meeting Reservation System is a system for allocation of resources (e.g. rooms and equipment) within specified time-slots and requesting participants for meetings or similar. Resources are defined as being any kind of resource with a set of properties related to it. Persons invited to the meeting should be automatically notified and requested for response.

- The reservation system should be able to present a list of reservation suggestions based on the requirements set by the organizer. Typical kinds of organizer requirements are: time period, duration, equipment, room capacity, equipment capacity and required participants.
- Notifications should provide efficient feedback to participants and it should be very simple to respond to them.
- The system should help the users making the most appropriate reservation by making suggestions based on input from the user as well as relevant information that is available. (E.g. suggest meeting room(s) nearby the requesting user, make suggestion based on room properties (number of sites, room equipment etc), check schedule of required participants and give intelligent suggestions and feedback to the user, suggest additional equipment if appropriate (e.g. extension lead, appropriate plugs (e.g. for power supply when there is an international meeting)).

3.5.2 Assumptions

The Meeting Reservation System modeled in this context are based on the following assumptions:

- Availability of an organization structure and information system with employee information.
- Meeting invitations are sent via e-mail (asynchronous).

- Allocation of resources is transacted on the server-side of the system (synchronous).
- The usage of the system is assumed to be internal within one organization structure. However, the models described, can be applied to virtual and/or collaborating organization structures.

3.6 Enterprise Viewpoint Specification

In the Enterprise Viewpoint Specification, we structure communities for the Meeting Reservation System. This includes describing the general structure of the communities, their enterprise objects, roles, the objectives of those roles, and the enterprise processes involved in accomplishing those objectives.

3.6.1 Community Structure

Figure 19 shows the community structure for the Meeting Reservation System. The top-level community that this system is targeted at is an Organization community. An organization can consist of several interacting departmental communities. Two sub communities of interest for the Meeting Reservation System are: The Project working community in which we find the end-users of the system, and the Administration community in which we find the system operators responsible for running and administrating the system.

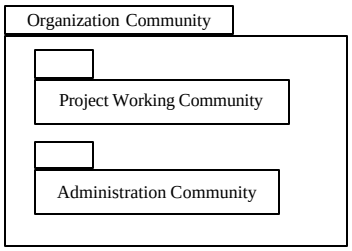


Figure 19: Organization Community

Figure 20 shows a “rich picture” describing the relationship among communities, roles in communities and objects performing those roles. (This is an ad-hoc UML diagram, where classes are used to represent objects and actors are used to represent roles. A role is performed by an object, which is shown using a dependency from an actor to a class.)

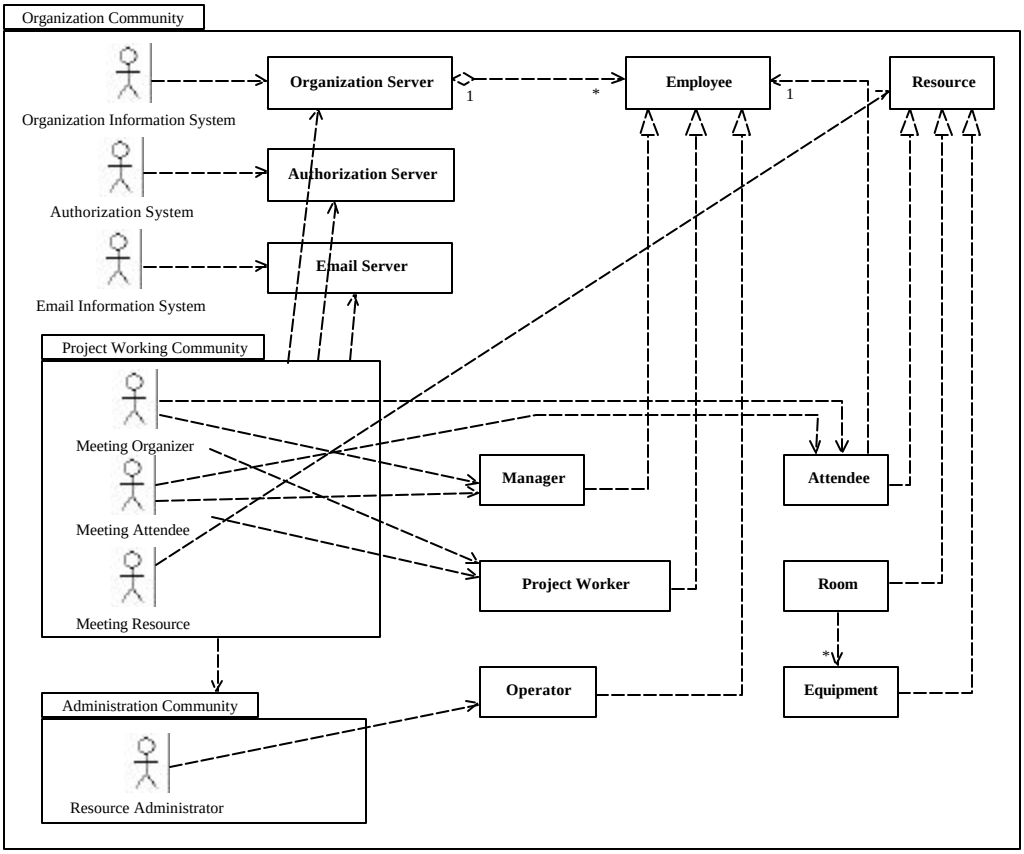


Figure 20: Communities, Enterprise Objects, and Roles

3.6.2 Objectives of each Community

The objectives of each community are shown below:

- The Project Working Community is responsible for accomplishing project activities, and is a sub community of the Organization Community.
- The Administration Community is responsible for supporting the other sub communities of the Organization Community.

3.6.3 The Project Working Community

The end-users of the Meeting Reservation System are primarily found in the Project Working Community.

3.6.3.1 Scope

There are many activities involved regarding the life cycle of a research or a development project. The system described here is restricted to a few supporting activities, namely:

- Planning project/research meetings

- Arranging and holding project/research meetings.

3.6.3.2 Enterprise Objects

Enterprise objects participating in this community and performing those roles described below are the following:

- Manager::Employee
- Project Worker::Employee
- Attendee::Resource
- Room::Resource
- Equipment::Resource
- Organization Server (existing system)
- Email Server (existing system)
- Authorization Server (existing system)

3.6.3.3 Roles

Figure 21 shows a use case diagram for the roles (actors) in the Project Working Community with regards to the Meeting Reservation System.

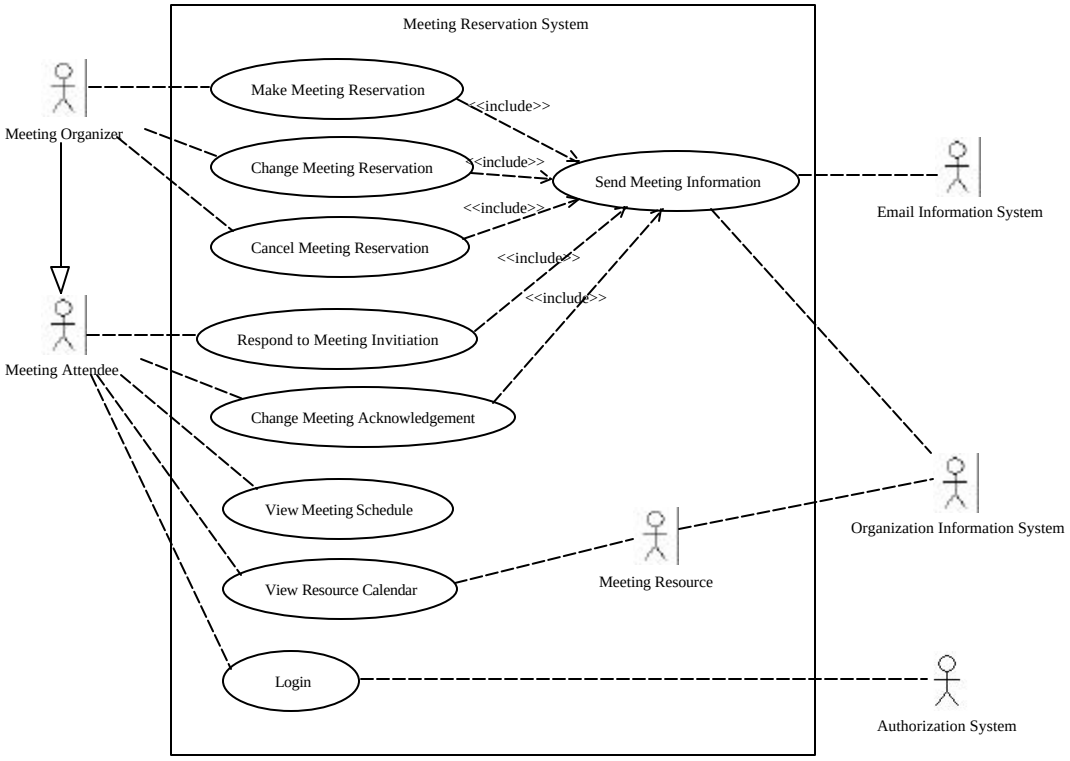


Figure 21: Project Working Community Use Case View

Detailed roles required for this community to function are the followings:

- Meeting Organizer (performed by ::Employee)
- Meeting Attendee (performed by ::Employee)
- Meeting Resource (performed by ::Resource)
- Organization Information System (performed by Organization Server)
- Email Information System (performed by Email Server)
- Authorization System (performed by Authorization Server)

3.6.3.4 Policies

Here are some policies (constraints) placed on various roles. Note that more constraints, such as pre-conditions, are described in Process section below.

- A Meeting Resource role of type Attendee must have an associated Employee object.
- The Organization Information System must have all Meeting Resource roles of type Attendee registered.

3.6.3.5 Processes

The processes for this community are described in terms of the Business Process Profile and corresponding EDOC notation.

Figure 22 shows a high-level diagram that describes the main process from the planning of a meeting until it is cancelled or is held. This process is further elaborated below in detailed diagrams for the two sub processes.

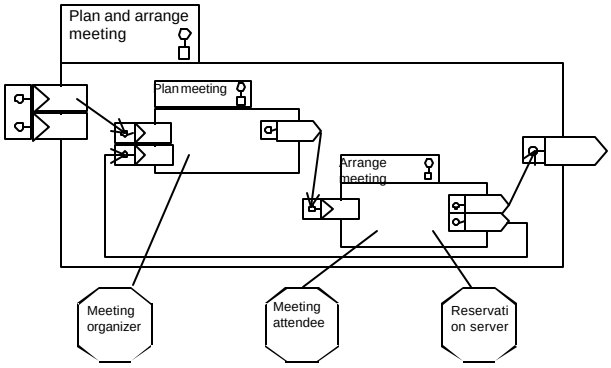


Figure 22: "Plan and arrange meeting" process

Figure 23 shows the details for the "Plan meeting" sub process.

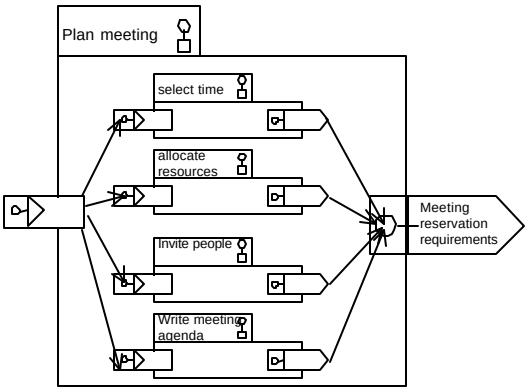


Figure 23: "Plan meeting" sub process details

Figure 24 shows the details for the "Arrange meeting" sub process. This sub process consists of several activities, some of which are described in own diagrams below.

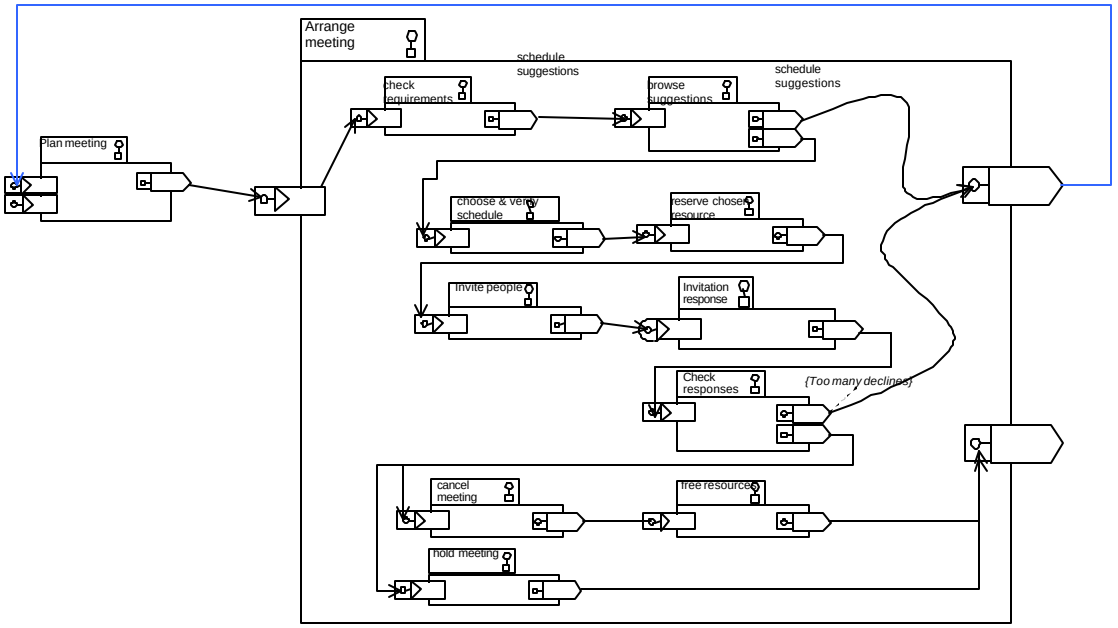


Figure 24: "Arrange meeting" sub process details

Figure 25 shows the details for the "Check requirements" activity.

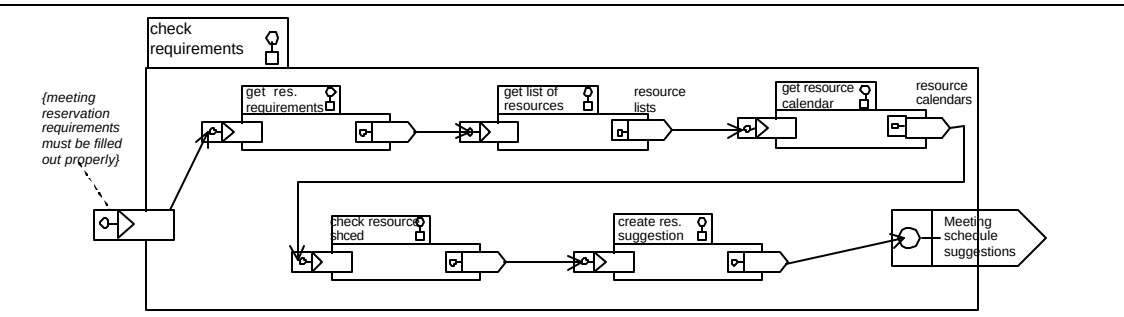


Figure 25: "Check requirements" activity details

Figure 26 shows the details for the "Reserve chosen resources" activity.

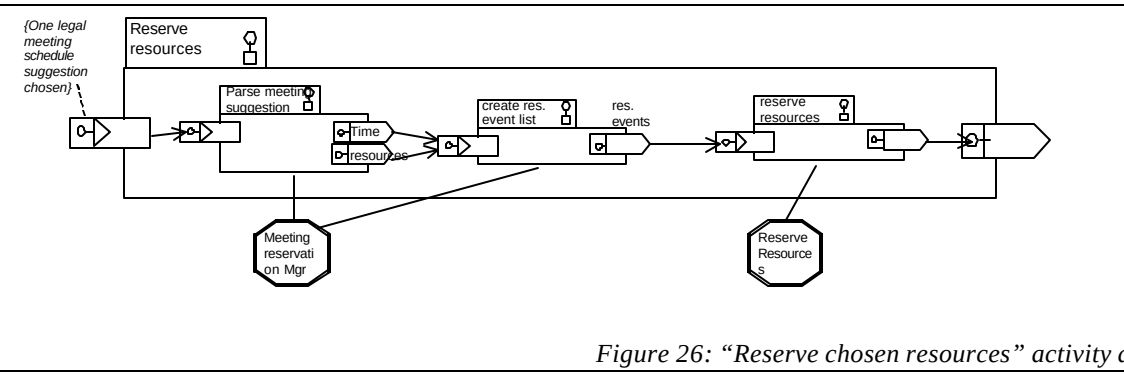


Figure 26: "Reserve chosen resources" activity details

3.6.3.6 Activity Specification

The activity specifications for this community are described using the Business Process Profile. The activities are derived from the process diagrams presented above.

Figure 27 shows the activity specification for the "Check requirements" activity.

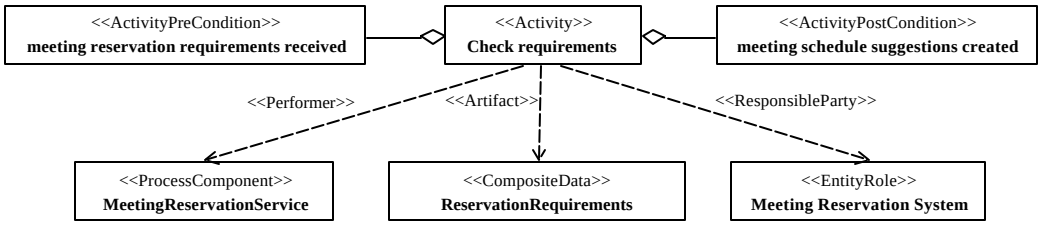


Figure 27: "Check requirements" activity specification

Figure 28 shows the activity specification for the "Reserve chosen resources" activity.

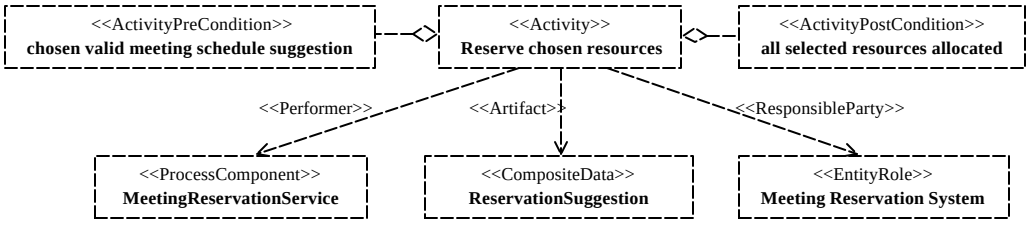


Figure 28: "Reserve chosen resources" activity specification

Figure 29 shows the activity specification for the "Respond to meeting invitation" activity.

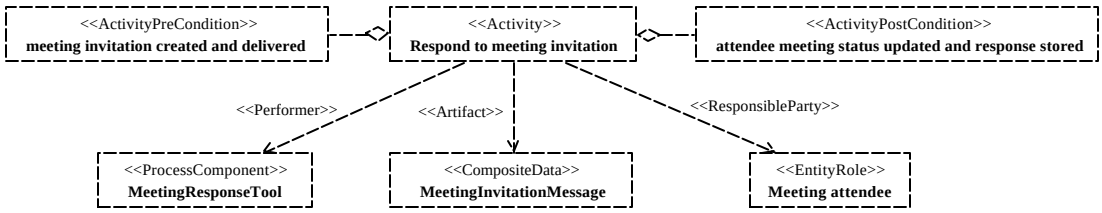


Figure 29: "Respond to meeting invitation" activity specification

3.6.4 The Administration Community

The administration community is responsible for the operations and maintenance information systems supporting the project working community.

3.6.4.1 Scope

The scope of the administration community described in this context is restricted to the operations and maintenance of the Meeting Reservation System.

3.6.4.2 Enterprise Objects

The enterprise objects participating in this community and performing the roles described below are the following:

- Operator::Employee
- Attendee::Resource
- Room::Resource
- Equipment::Resource
- Organization Server (existing system)
- Email Server (existing system)
- Authorization Server (existing system)

3.6.4.3 Roles

Figure 30 shows a use case model for the roles (actors) in the Administration community with regards to the Meeting Reservation System.

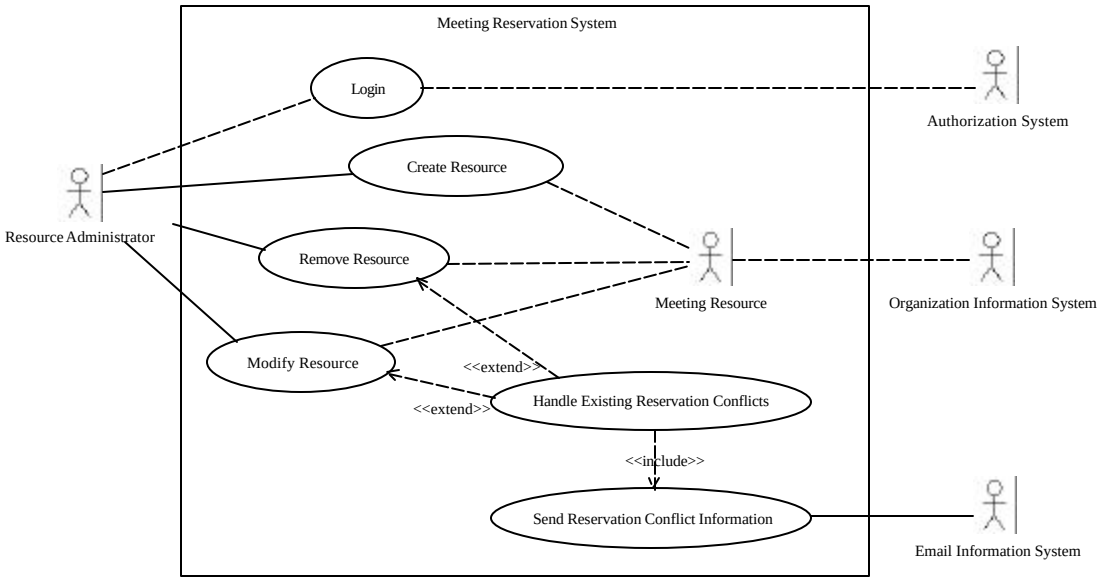


Figure 30: Administration Community Use Case View

Detailed roles required for this community to function are the followings:

- Resource Administrator (performed by Operator::Employee)
- Meeting Resource (performed by ::Resource)
- Organization Information System (performed by Organization Server)
- Email Information System (performed by Email Server)
- Authorization System (performed by Authorization Server)

3.6.4.4 Policies

Here are some policies (constraints) placed on various roles. Note that more constraints, such as pre-conditions, are described in the process section below.

- A Meeting Resource role of type Room or Equipment must have an associated real-life, physical object.

3.6.4.5 Processes

The processes for this community are described in terms of the Business Process Profile and corresponding EDOC notation.

Figure 31 shows a high-level diagram that describes the main process of resource administration.

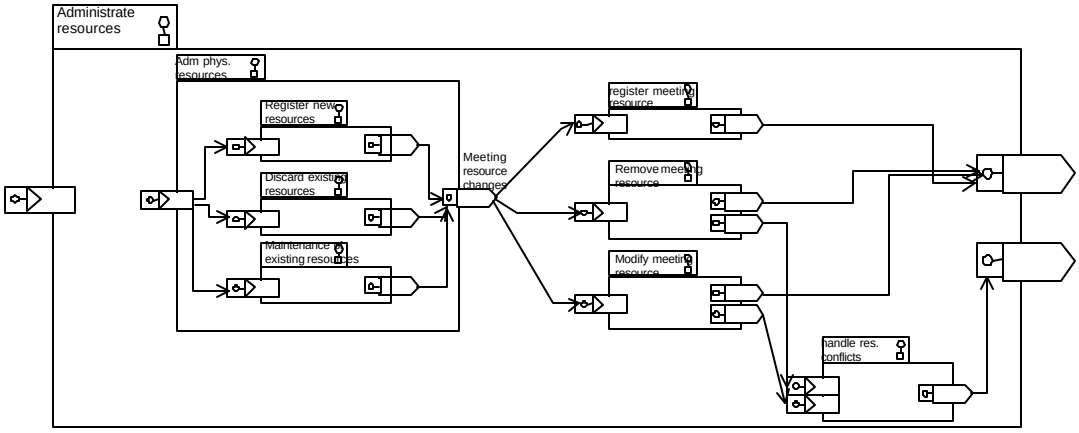


Figure 31: "Administrate resources" process

The remaining process details of the administration community are not described here.

3.6.4.6 Activity Specification

The activity specifications for this community are described using the Business Process Profile. The activities are derived from the process diagrams presented above.

Figure 32 shows the activity specification for the "Remove meeting resource" activity.

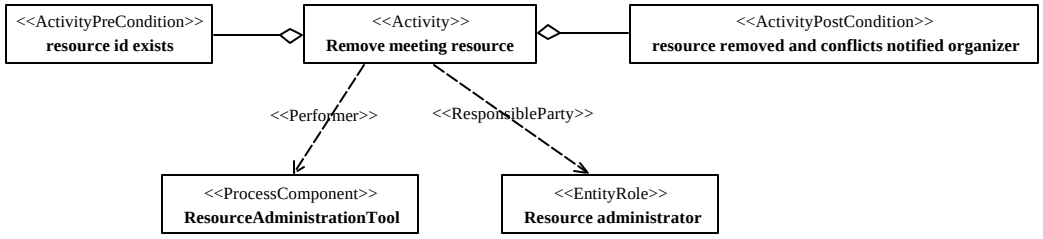


Figure 32: "Remove meeting resource" activity specification

3.7 Information Viewpoint

The information viewpoint describes the information context of the Meeting Reservation System using the Entity and Relationship Profile.

3.7.1 Server-Side Information View

Figure 33 shows the server-side information view that describes the information context represented on the server tier.

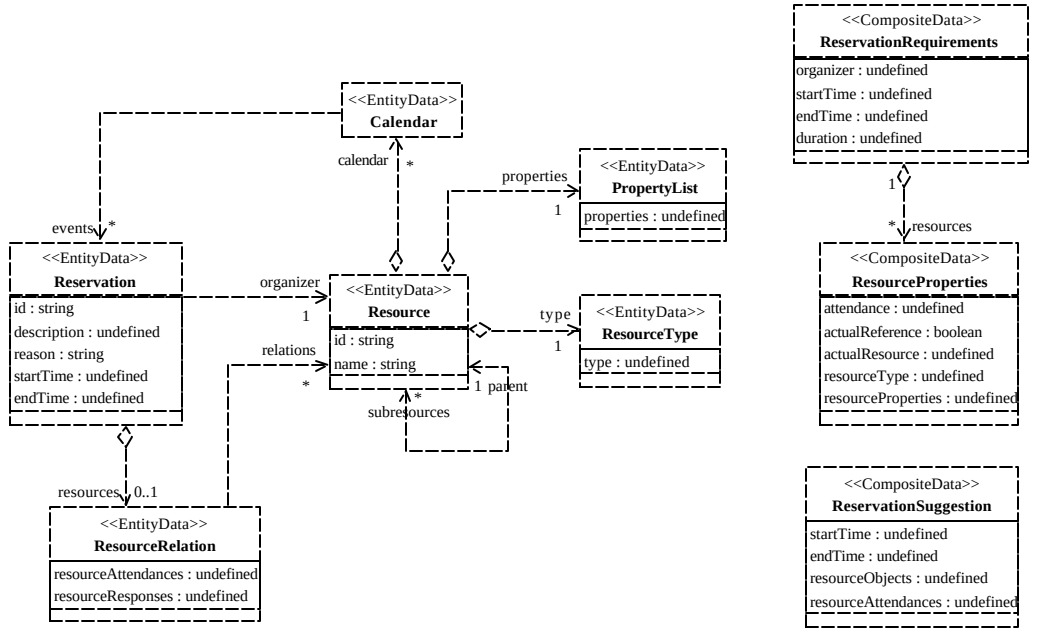


Figure 33: Server-Side Information View

Each of the information profile types are described in more detail below:

- *Reservation* represents the allocation of a set of resources for a specific time frame. A *Reservation* has a resources role of type *ResourceRelation* and an organizer role of type *Resource*.

- *ResourceRelation* has a relations role of type *Resource* that represents the set of allocated resources. A *ResourceRelation* also has information about resource attendances and responses.

- *Resource* represents the target for reservations. A *Resource* has a *Calendar* that contains the set of reservations it participates in. A *Resource* can have subresources role and a parent role of type *Resource* describing the recursive structure of a resource. A *Resource* has properties defined in a *PropertyList* and a type defined in a *ResourceType*.

- *Calendar* represents a plan for resources. A *Calendar* defines a set of events of type *Reservation*.

- *PropertyList* represents a set of defined properties for a resource.

- *ResourceType* represents the resource type (e.g. attendee, room, equipment).

- *ReservationRequirements* is a composite data element representing a composed requirement specification for a reservation. It contains a set of *ResourceProperties.*

- *ResourceProperties* control required settings a reservation. *ResourceProperties* can be used to refer to an actual resource or describe the properties of a suggested resource.

- *ReservationSuggestion* represents the suggested resources for a reservation.

Figure 34 describes the two entity components derived from the information view.

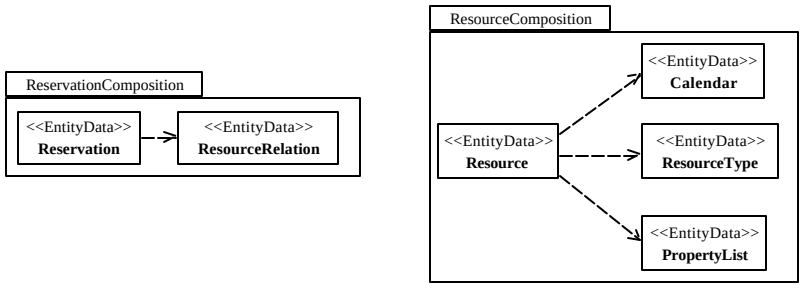


Figure 34: Server-Side Composition View

The entities corresponding to the ReservationComposition and ResourceComposition are further elaborated in the Computational Viewpoint.

3.7.2 Client-Side Information View

Figure 35 shows the client-side information view that describes the information context on the client tier. The server-side information view typically supports a more generic information model, while the client-side information view represents a local view, possibly augmented with information objects necessary to support business logic near the client tier.

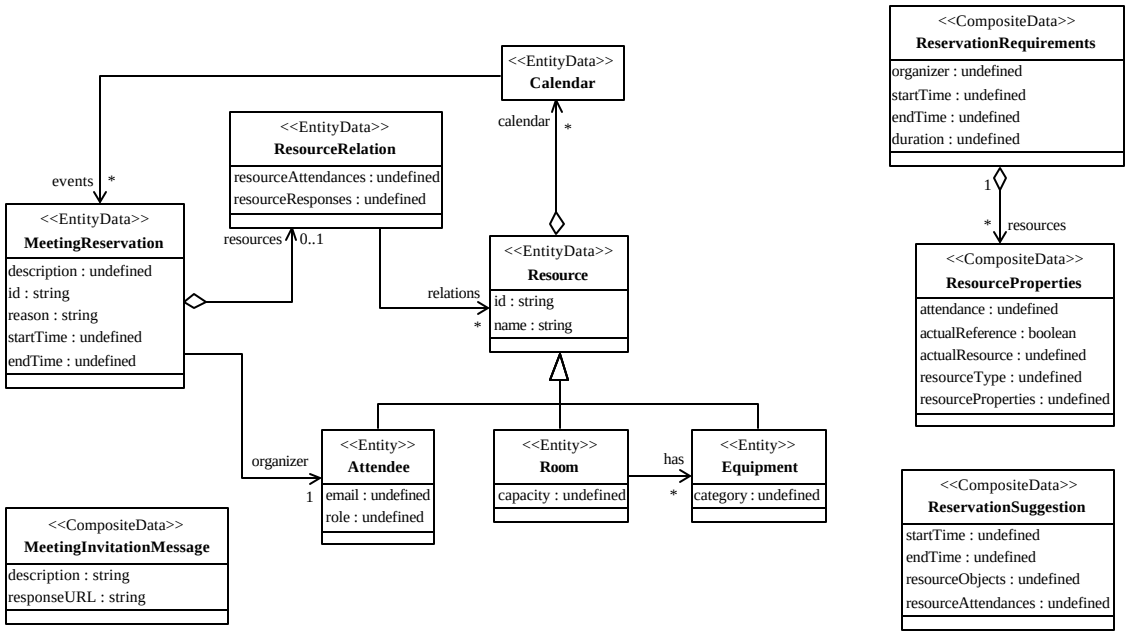


Figure 35: Client-Side Information View

Each of the information profile types are described in more detail below:

- *MeetingReservation* represents the local view of a reservation.
- *Attendee* represents the local view of a resource, with defined properties, of type "Attendee".
- *Room* represents the local view of a resource, with defined properties, of type "Room".
- *Equipment* represents the local view of a resource, with defined properties, of type "Equipment".
- *MeetingInvitationMessage* is a composite data element that is sent (via e-mail) to every invited attendee containing an URL that is used to start a meeting response tool.

The remaining information elements are as described for the server-side information view.

No compositions are described on the client-side since the entity data are to be interpreted as object by value data controlled in a local, client workspace session

3.8 Computational Viewpoint

The Computation Viewpoint is mainly described using the Component Collaboration Architecture (CCA) profile. In the Computational Viewpoint Specification, Computational Objects are derived and presented as CCA Process Components. Port specification as interface specifications for comp utational object, and Protocol specification as interaction specifications between computational objects are described.

3.8.1 Overview

Figure 36 shows an overview of the component structure model, which describes the relationships between the *Tool*, *Service*, and *Manager* component types used to model the Meeting Reservation System.

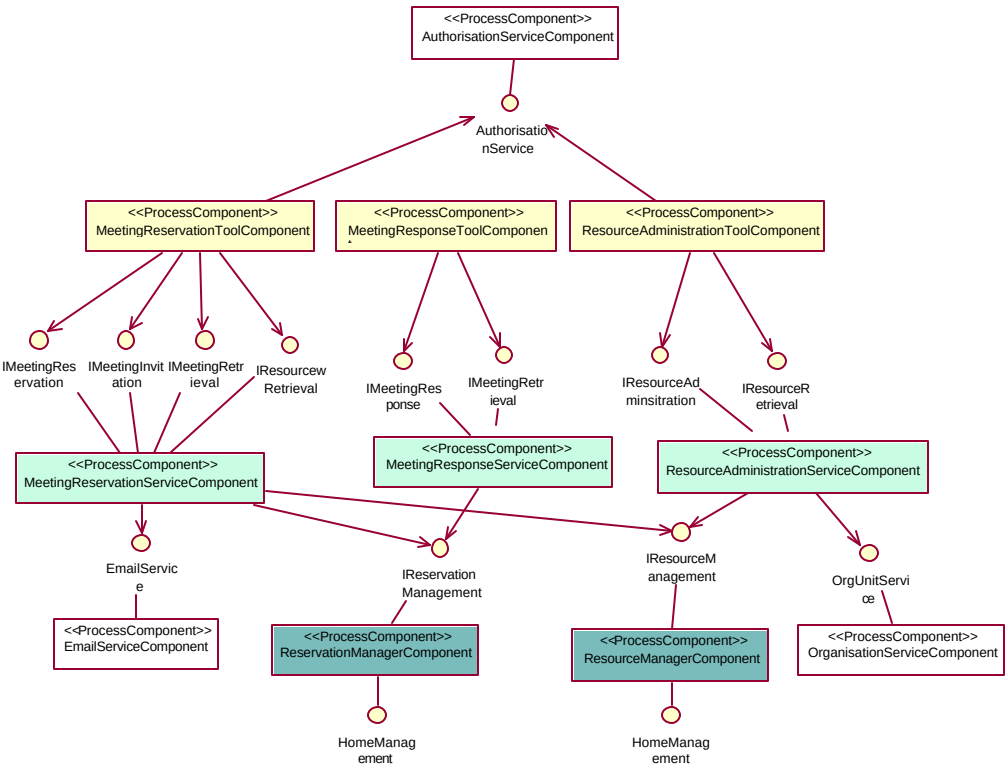


Figure 36: Component Structure Overview

3.8.2 Identified set of Legacy Wrapper Service Components

The Organization Community contained three existing systems (Organization, Email, and Authorization server) that we can view as “classical” services having interfaces, which exposes their usage. In order for the Meeting Reservation System to be able to interact with each of these services, wrapper process components are needed:

- OrganizationServiceComponent of type ProcessComponent
- EmailServiceComponent of type ProcessComponent
- AuthorizationServiceComponent of type ProcessComponent

Each of these components is elaborated in diagrams below. The wrapper process components defined below can be replaced by the actual server systems if they provide the interfaces described.

Figure 37 shows the component structure for the organization service component.

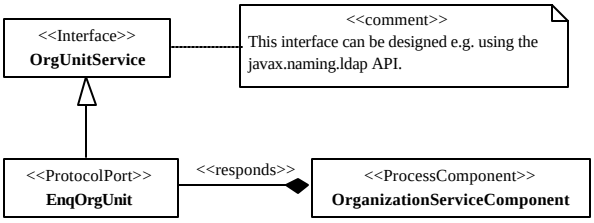


Figure 37: Organization Service Component Structure

Figure 38 shows the component structure for the email service component.

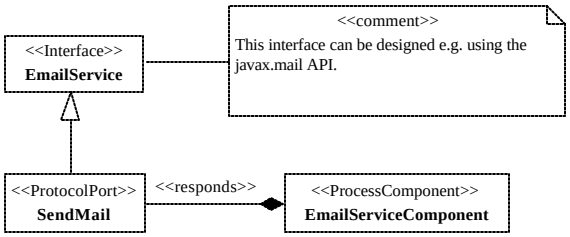


Figure 38: Email Service Component Structure

Figure 39 shows the component structure for the authorization service component.

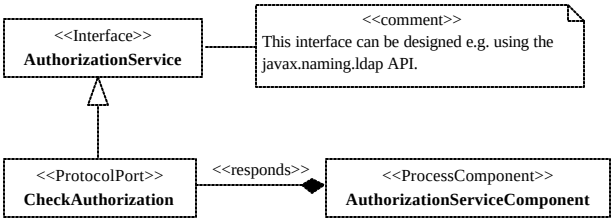


Figure 39: Authorization Service Component Structure

3.8.3 Identified set of Entity Components

In the Information Viewpoint we defined two entity compositions (ReservationComposition and ResourceComposition) that we now map onto corresponding entity components:

- ReservationComponent of type Entity
- ResourceComponent of type Entity

3.8.3.1 Reservation Component

Figure 40 shows the entity view for the reservation entity component.

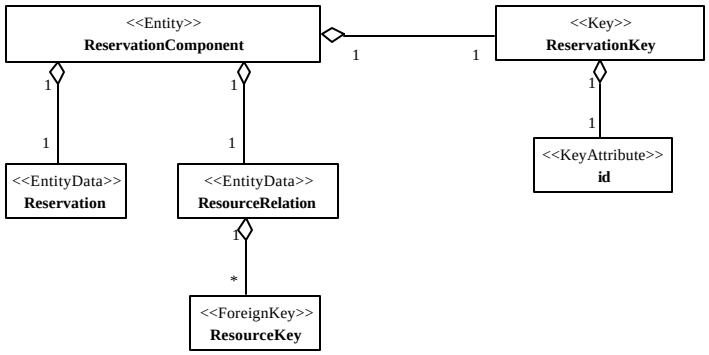


Figure 40: Reservation Entity Component Entity View

Figure 41 shows the component structure for the reservation entity component.

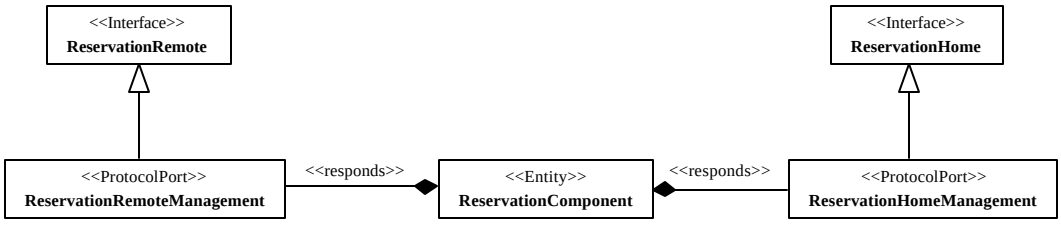


Figure 41: Reservation Entity Component Structure

Figure 42 shows the protocol structure for the ReservationRemote interface (protocol) describing the operations defined.

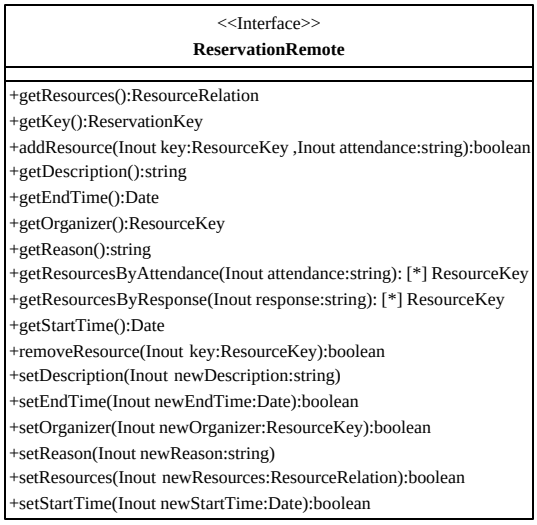


Figure 42: ReservationRemote Interface Structure

3.8.3.2 Resource Component

Figure 43 shows the entity view for the resource entity component.

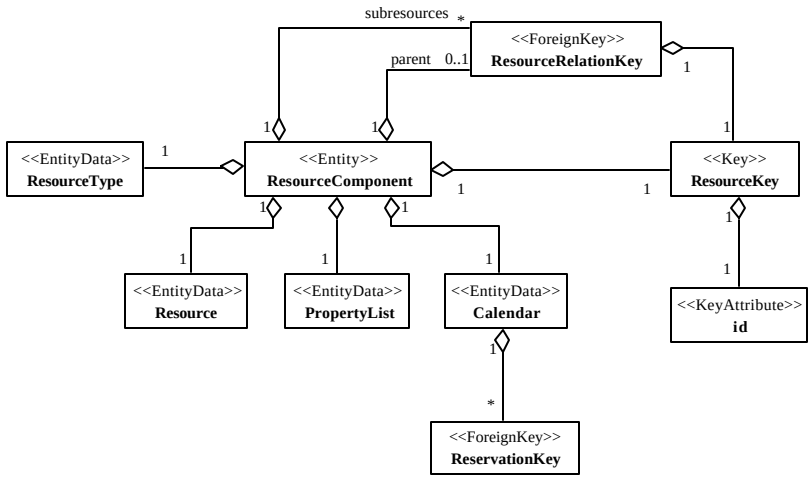


Figure 43: Resource Entity Component Entity View

Figure 44 shows the component structure for the resource entity component.

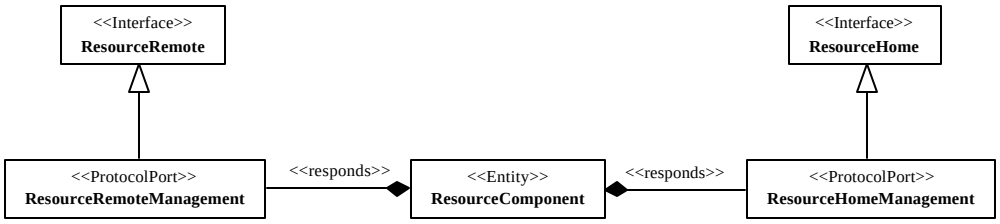


Figure 44: Resource Entity Component Structure

Figure 45 shows the protocol structure for the ResourceRemote interface (protocol) describing the operations defined.

Figure 45: ResourceRemote Interface Structure

3.8.4 Identified set of Computational Components

Other computational objects that are derived from the Enterprise Viewpoint specification and Information Viewpoint specification are (all of type ProcessComponent):

- MeetingReservationToolComponent: A component that represents the client application used for booking meetings.
- MeetingReservationServiceComponent: A client-side component that is used by the MeetingReservationTool.
- MeetingResponseToolComponent: A component that represents the client application used for responding to meeting invitations.
- MeetingResponseServiceComponent: A client-side component that is used by the MeetingResponseTool.
- ResourceAdministrationToolComponent: A component that represents the client application used for administrating resources.
- ResourceAdministrationServiceComponent: A client-side component that is used by the ResourceAdministrationTool.
- ReservationManagerComponent: A server-side component that manages reservations.
- ResourceManagerComponent: A server-side component that manages resources.

The specifications of the tool, service and manager components are elaborated below. The specification are structured according to a four tier architecture. The tool and service components are client-side components. The manager and entity components are server-side components. Interaction between the client and server side is enforced using only the service components.

3.8.4.1 Meeting Reservation Tool Component

Figure 46 shows the component structure for the meeting reservation tool component.

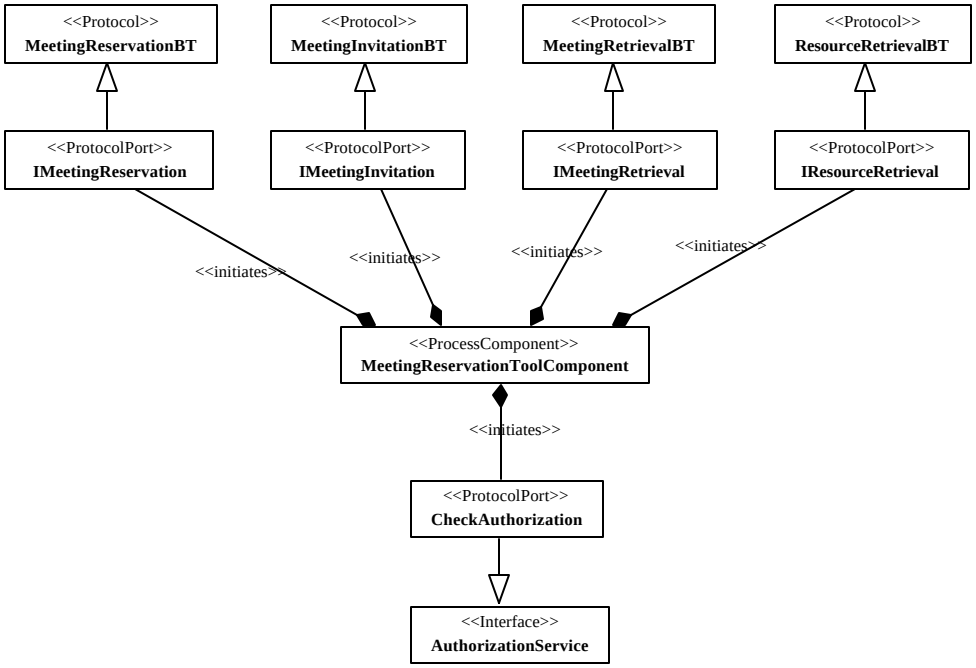


Figure 46: Meeting Reservation Tool Component Structure

3.8.4.2 Meeting Reservation Service Component

Figure 47 shows the component structure for the meeting reservation service component.

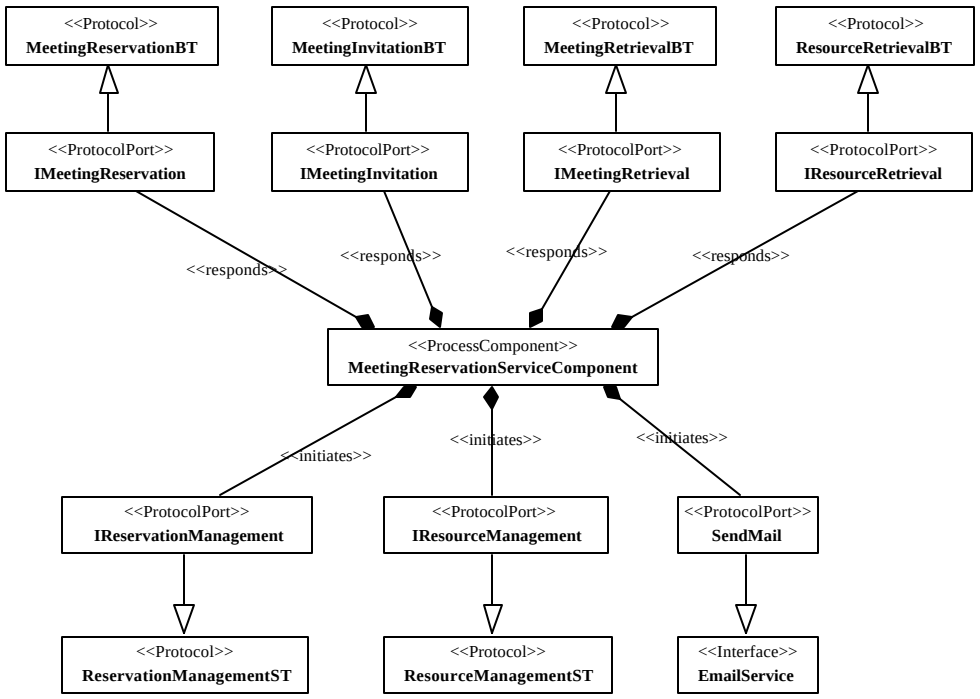


Figure 47: Meeting Reservation Service Component Structure

3.8.4.3 Meeting Response Tool Component

Figure 48 shows the component structure for the meeting response tool component.

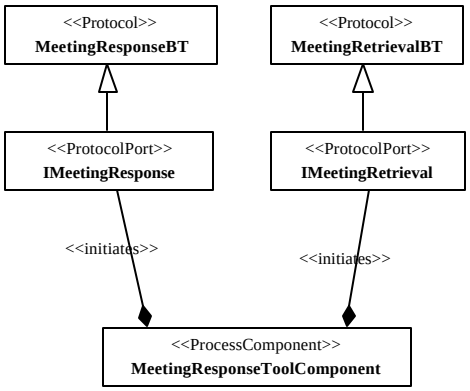


Figure 48: Meeting Response Tool Component Structure

3.8.4.4 Meeting Response Service Component

Figure 49 shows the component structure the meeting response service component.

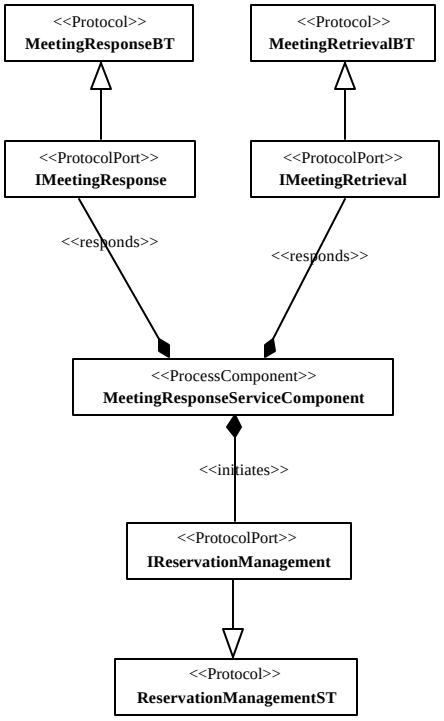


Figure 49: Meeting Response Service Component Structure

3.8.4.5 Resource Administration Tool Component

Figure 50 shows the component structure for the resource administration tool component.

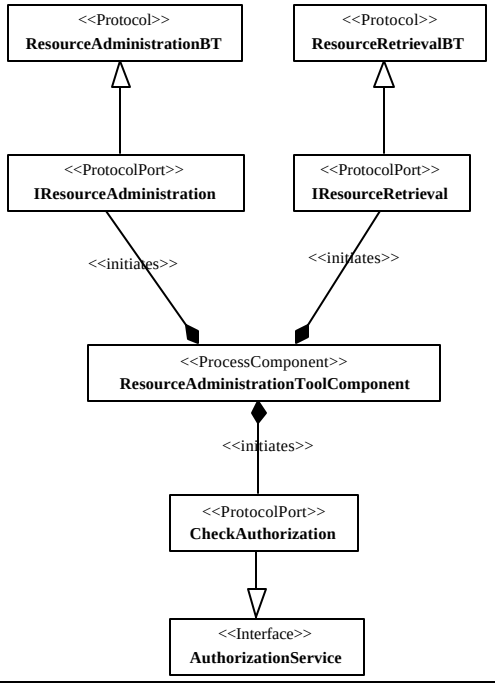


Figure 50: Resource Administration Tool Component Structure

3.8.4.6 Resource Administration Service Component

Figure 51 shows the component structure for the resource administration service component.

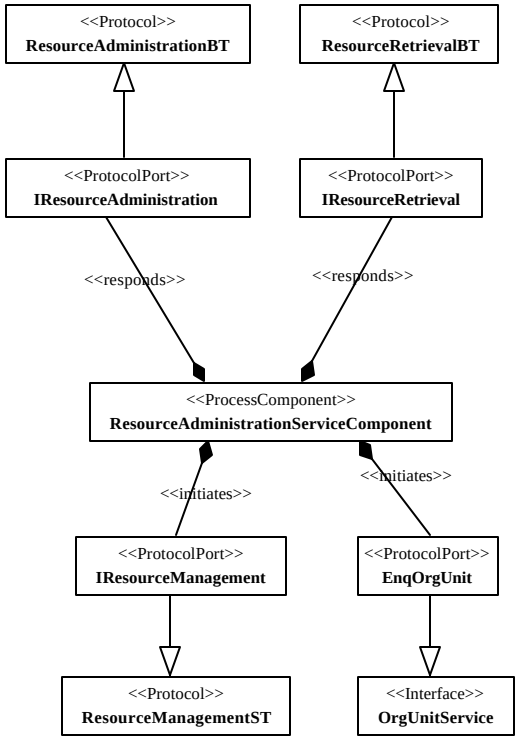


Figure 51: Resource Administration Service Component Structure

3.8.4.7 Reservation Manager Component

Figure 52 shows the component structure for the reservation manager component.

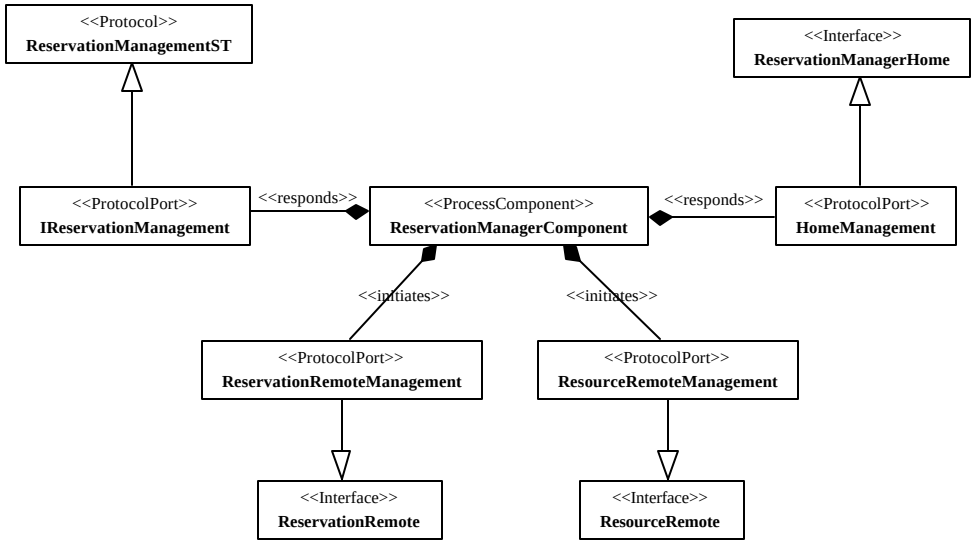


Figure 52: Reservation Manager Component Structure

3.8.4.8 Resource Manager Component

Figure 53 shows the component structure for the resource manager component.

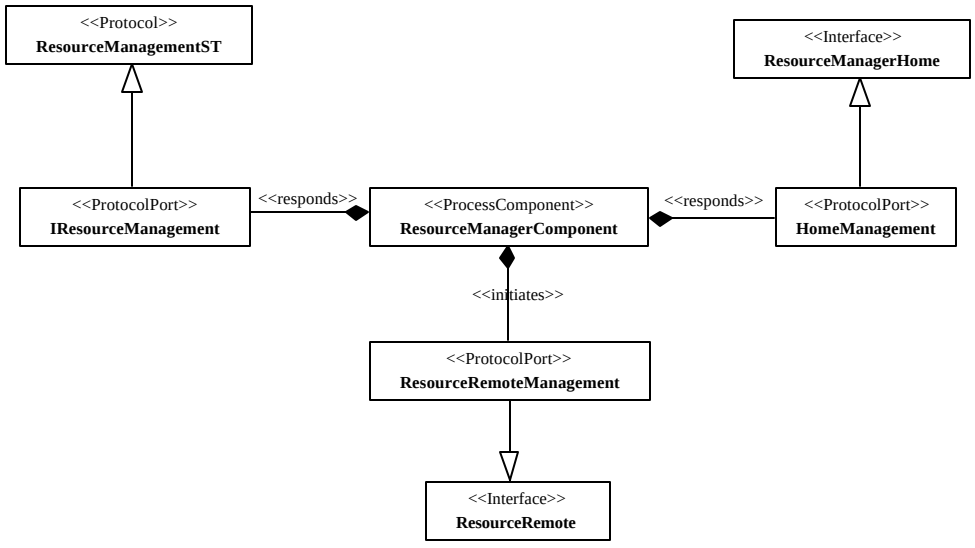


Figure 53: Resource Manager Component Structure

3.8.5 Protocol Specification

The following are partial protocol specifications for some identified business transactions using CCA.

3.8.5.1 Meeting Invitation Protocol

Figure 54 shows an activity diagram that describes the protocol for the meeting invitation business transaction.

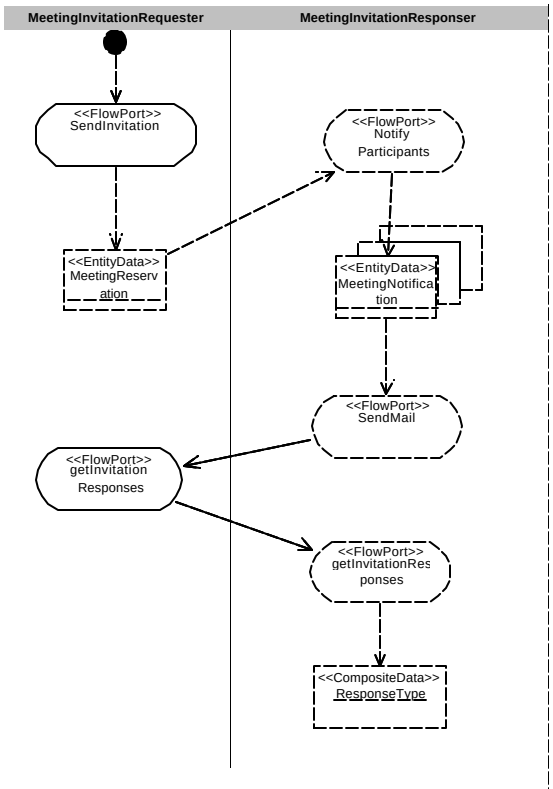


Figure 54: Meeting Invitation Protocol

Figure 55 shows the protocol structure for the meeting invitation business transaction. Since meeting invitations and responses can be sent asynchronously via e-mail, publisher and flow ports are defined that supports this business transaction.

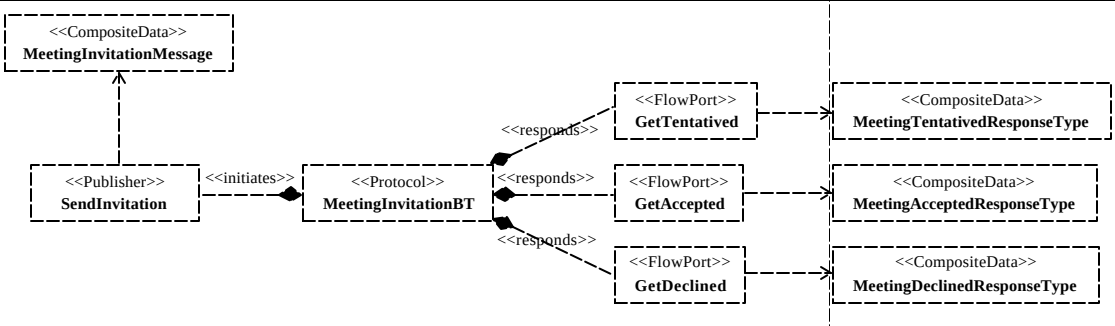


Figure 55: Meeting Invitation Business Transaction Protocol Structure

Figure 56 shows the protocol choreography for the meeting invitation business transaction.

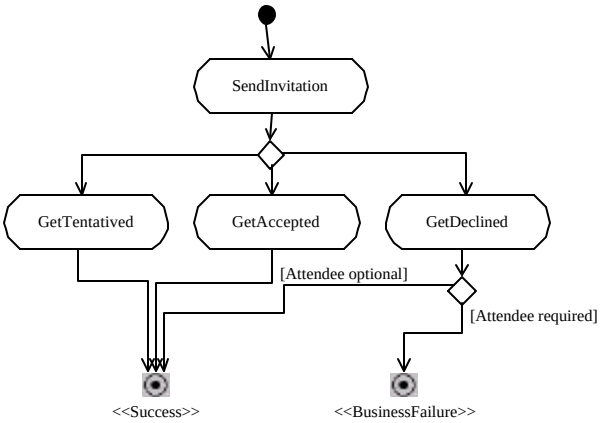


Figure 56: Meeting Invitation Business Transaction Protocol Choreography

3.8.5.2 Meeting Reservation Protocol

Figure 57 shows an activity diagram that describes the protocol for the meeting reservation business transaction.

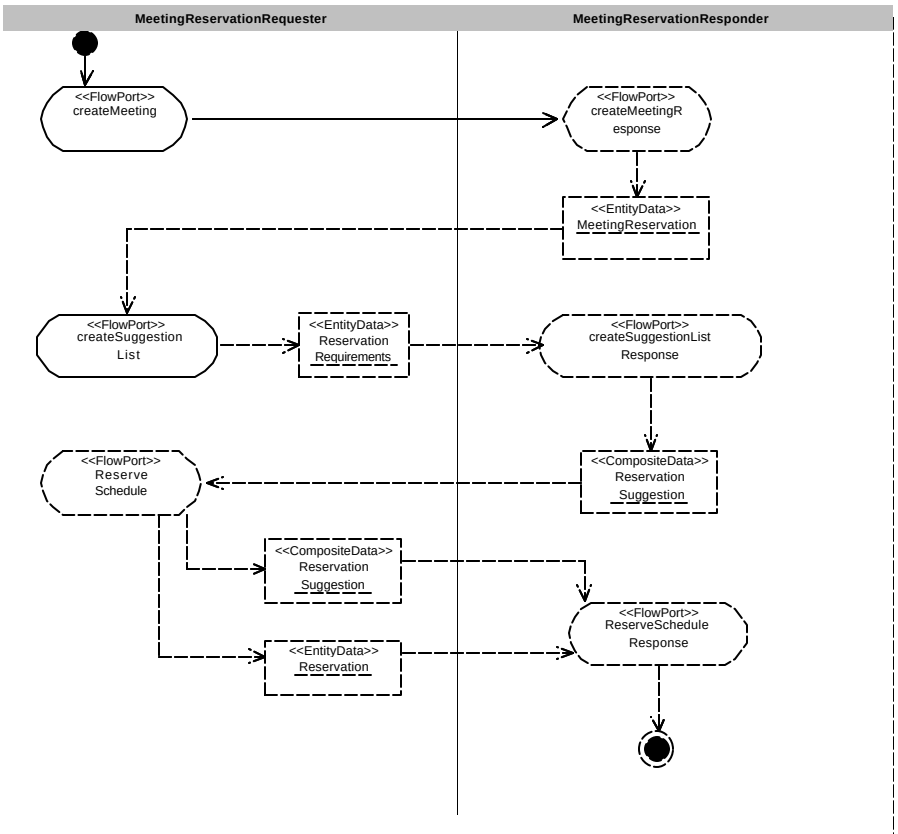


Figure 57: Meeting Reservation Protocol

Figure 58 shows the protocol structure for the meeting reservation business transaction.

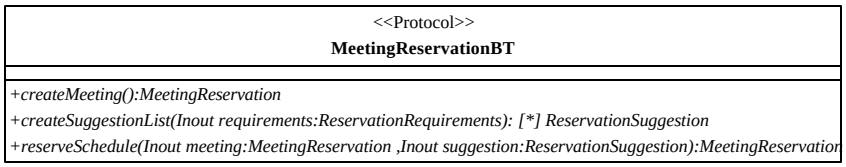


Figure 58: Meeting Reservation Business Transaction Protocol Structure

3.8.5.3 Reservation Management Protocol

Figure 59 shows an activity diagram that describes the protocol for the reservation management system transaction.

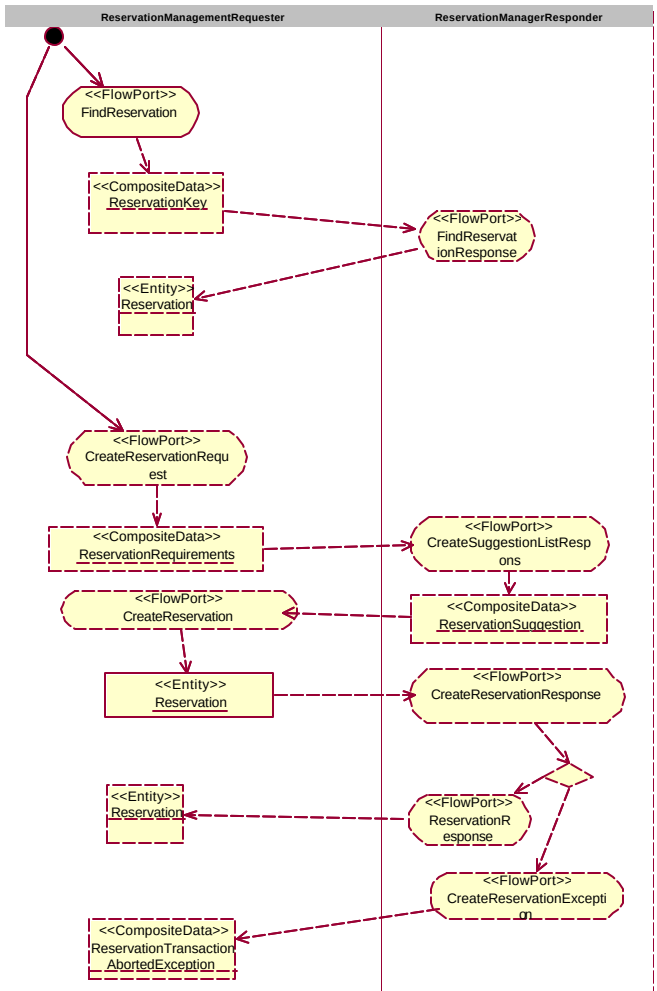


Figure 59: Reservation Management Protocol

Figure 60 shows the protocol structure for the reservation management system transaction.

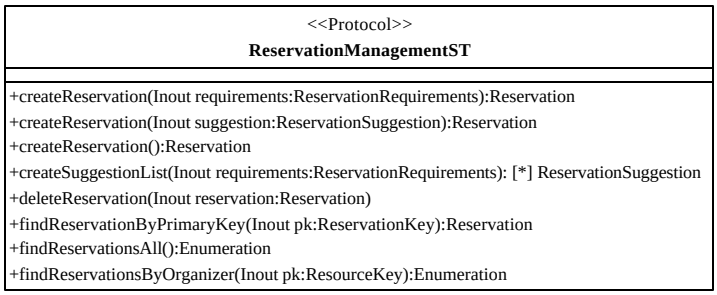


Figure 60: Reservation Management System Transaction Protocol Structure

3.8.6 Component Collaboration

Figure 61 shows a diagram describing the collaboration of some of the component specified above.

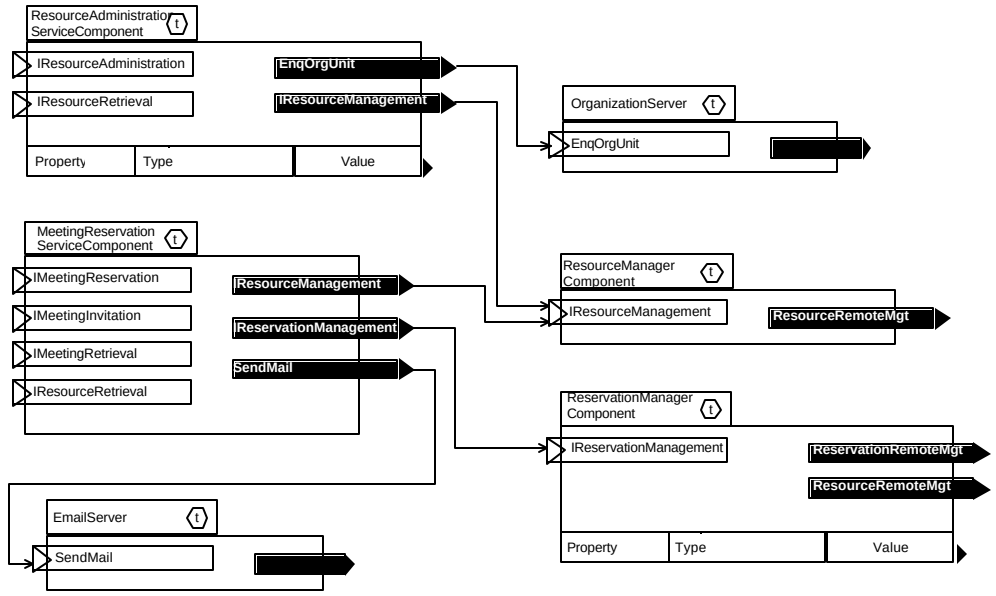


Figure 61: CCA Component Collaboration Model

3.9 Engineering Viewpoint Specification

The table, below, provides a mapping of the component elements specified in the Computational Viewpoint and the component elements implemented in the Technology Viewpoint below.

Computational Viewpoint Element	**Technology Viewpoint Element**
<<ProcessComponent>> OrganizationServiceComponent	not described

<<ProcessComponent>> EmailServiceComponent	not described
<<ProcessComponent>> AuthorizationServiceComponent	not described
<<Entity>> Reservation	<<EJBImplementation>> ReservationBean
<<Entity>> Resource	<<EJBImplementation>> ResourceBean
<<ProcessComponent>> MeetingReservationToolComponent	<<Java application>> MeetingReservationToolComponent
<<ProcessComponent>> MeetingReservationService	<<Java interface>> MeetingReservationService <<Java class>> MeetingReservationServiceImpl
<<ProcessComponent>> MeetingResponseToolComponent	<<Java application>> MeetingResponseToolComponent
<<ProcessComponent>> MeetingResponseService	<<Java interface>> MeetingReservationService <<Java class>> MeetingReservationServiceImpl
<<ProcessComponent>> ResourceAdministrationTool	<<Java application>> ResourceAdministrationTool
<<ProcessComponent>> ResourceAdministrationService	<<Java interface>> ResourceAdministrationService <<Java class> ResourceAdministrationServiceImpl
<<ProcessComponent>> ReservationManagerComponent	<<EJBImplementation>> ReservationManagerBean
<<ProcessComponent>> ResourceManagerComponent	<<EJBImplementation>> ResourceManagerBean

Further elaboration of the engineering viewpoint is not considered here.

3.10 Technology Viewpoint Specification

The technology viewpoint specification shows the J2EE implementation models for the components specified in the computational viewpoint.

3.10.1 Client-Side Components (Java models)

3.10.1.1 Meeting Reservation Tool and Service

Figure 62 shows the Java model for the meeting reservation tool and service components.

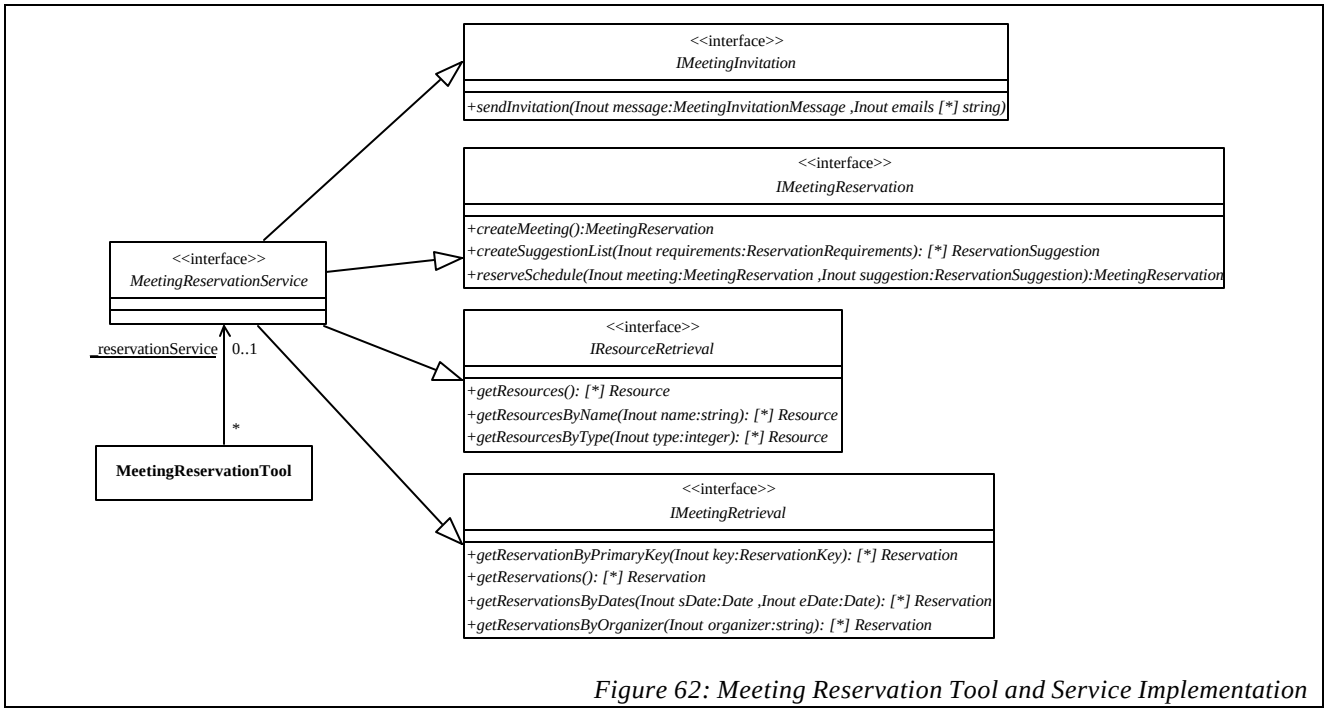


Figure 62: Meeting Reservation Tool and Service Implementation

3.10.1.2 Meeting Response Tool and Service

Figure 63 shows the Java model for the meeting response tool and service components.

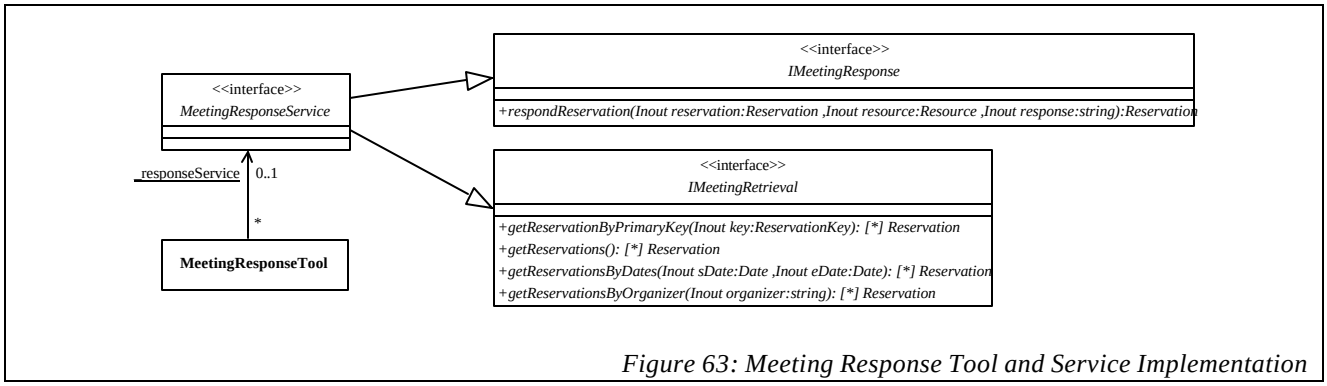


Figure 63: Meeting Response Tool and Service Implementation

3.10.1.3 Resource Administration Tool and Service

Figure 64 shows the Java model for the administration tool and service components.

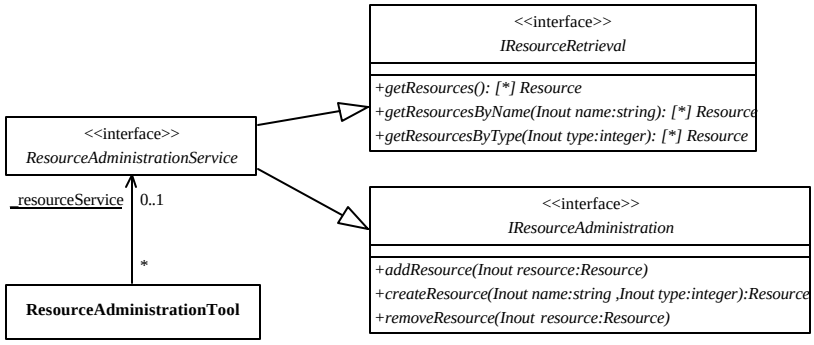


Figure 64: Resource Administration Tool and Service Implementation

3.10.2 Server-Side Components (EJB models)

3.10.2.1 Reservation Manager

Figure 65 shows the EJB model for the reservation manager component.

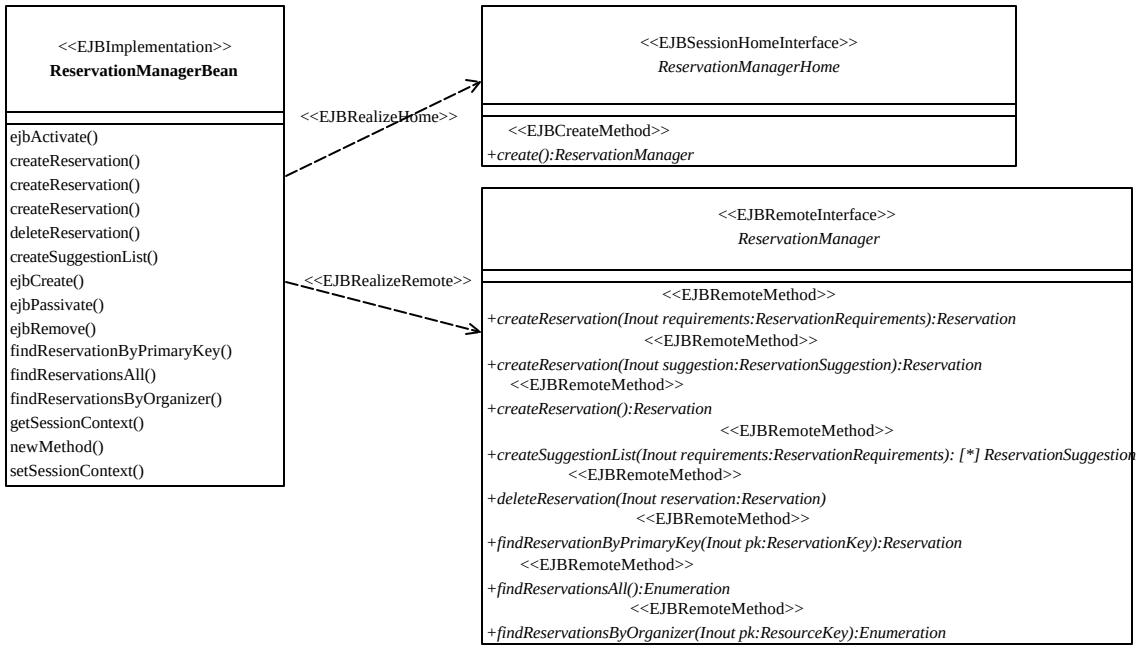


Figure 65: Reservation Manager Implementation

3.10.2.2 Resource Manager

Figure 66 shows the EJB model for the resource manager component.

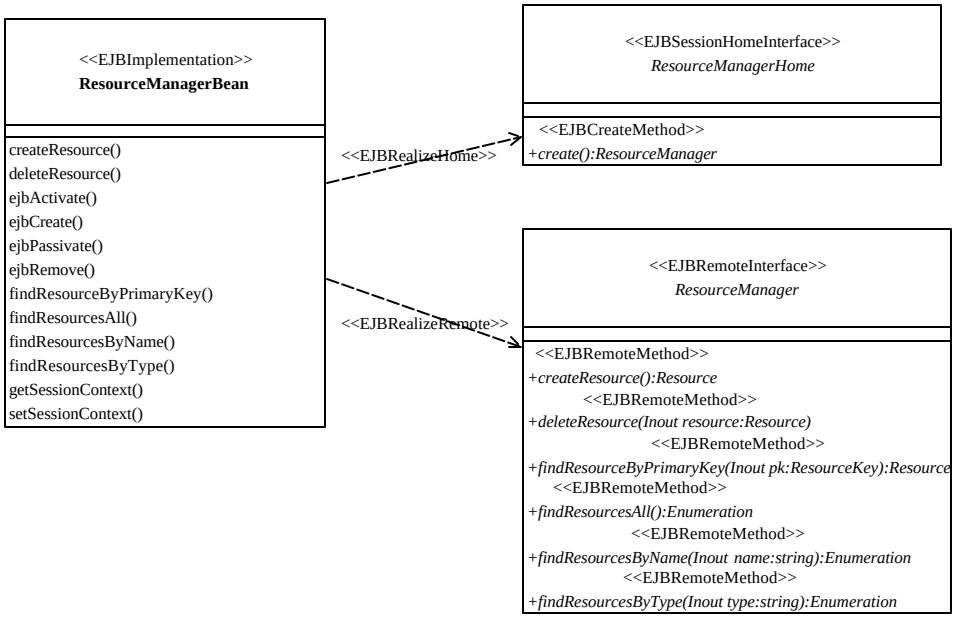


Figure 66: Resource Manager Implementation

3.10.2.3 Reservation Entity

Figure 67 shows the EJB model for the reservation entity component.

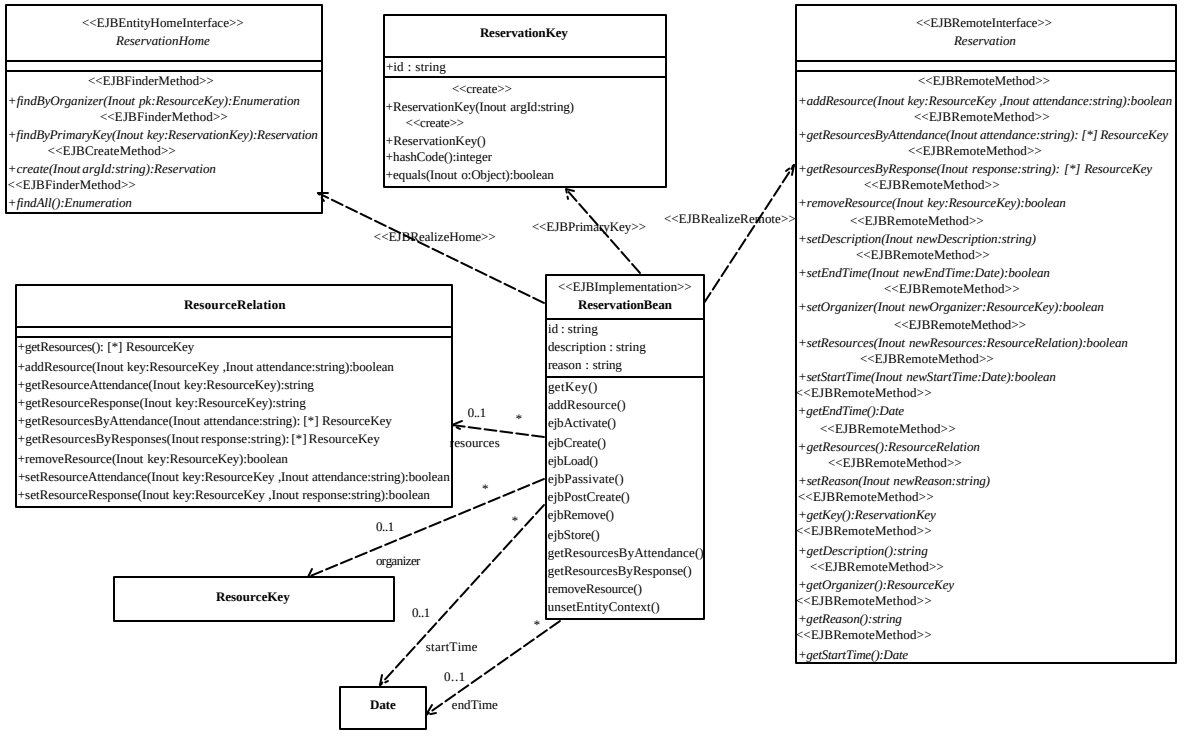


Figure 67: Reservation Entity Implementation

3.10.2.4 Resource Entity

Figure 68 shows the EJB model for the resource entity component.

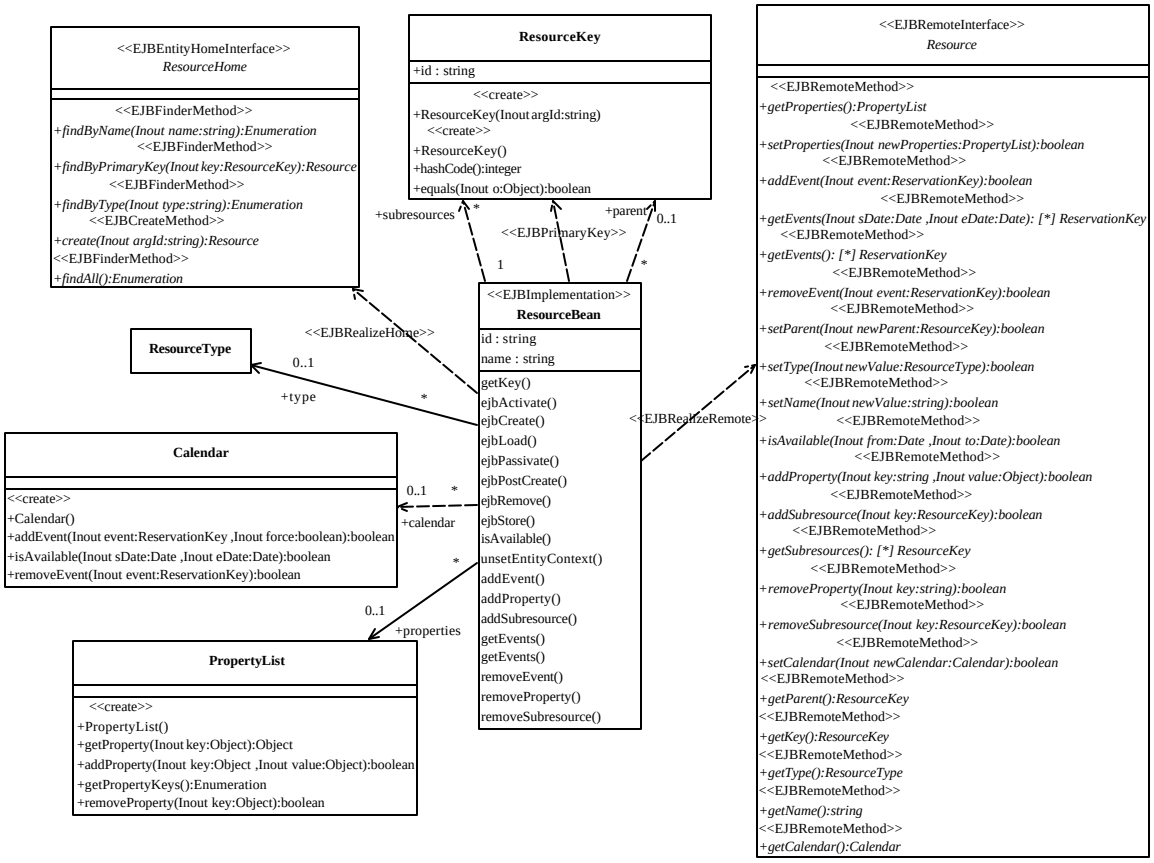


Figure 68: Resource Entity Implementation

Annex C - Example - Hospital Information System

Contents

List of Figures

4. Introduction

This annex describes the Radiological Community subset of a hospital information system model in terms of the UML Profile for EDOC. The annex uses the UML Profile to specify the Enterprise Viewpoint Specification, Information Viewpoint Specification, and Computational Viewpoint Specification for the subset.

4.1 Description

The model for the hospital information system is taken from the Hospital Information Reference Enterprise Model project in Japan. The purpose of the project is to provide a robust starting point for the design of hospital information ODP systems, using the concepts and rules defined in RM-ODP and ODP Enterprise Viewpoint Language, as well as using UML and the UML Profile for EDOC.

Since healthcare services are legislation-bound and culture-bound, this model includes some legislative and cultural requirements.

4.2 Assumptions of the hospital model

The model assumes that the hospital is a major regional hospital in Japan with approximately 300 beds. The model also makes the following assumptions.

- The hospital is not a postgraduate educational institution (no resident physicians are on the staff).
- The hospital provides no advanced specialty care such as that provided at university hospitals. Advanced specialty care includes renal dialysis, radiotherapy, etc.
- The hospital is a general hospital, i.e.,
  - The hospital is an Insurance Medical Facility (a hospital is accredited by a municipal governor to offer medical services under the public medical insurance scheme. Almost all the hospitals in Japan are Insurance Medical Facilities).
  - The hospital has no dental department (no dentists are on the staff).
  - The hospital is not involved in clinical trials.
  - The hospital has no surgery department, emergency department, or nutrition department.

5. Enterprise Viewpoint Specification

The Enterprise Viewpoint Specification specifies the structures of communities first. The top-level community called the Hospital Community is divided into sub-communities. This section describes the overall structure including communities, objects, and roles.

5.1 Overview

5.1.1 Community structure

Figure 1 shows the community structure for the hospital model, and enterprise objects. The top-level community is called the Hospital Community and is composed of several interacting departmental communities. Two of the interacting departmental communities, Patient Care and Administration, are further decomposed into their sub-communities

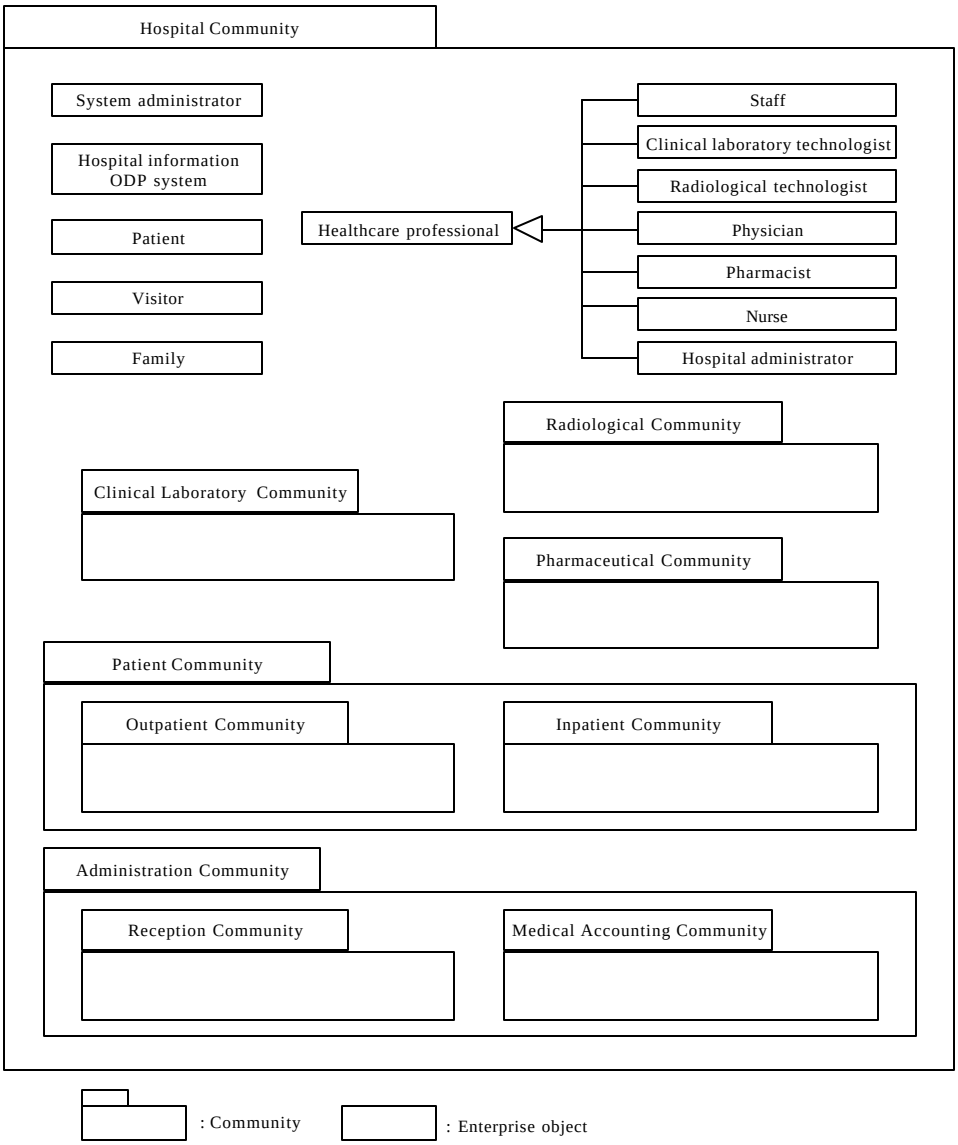


Figure 1: Hospital community

The following figures represent the relationship of the communities, the roles in the communities, and the objects performing the roles. This figure also identifies which objects are to be shared by multiple communities

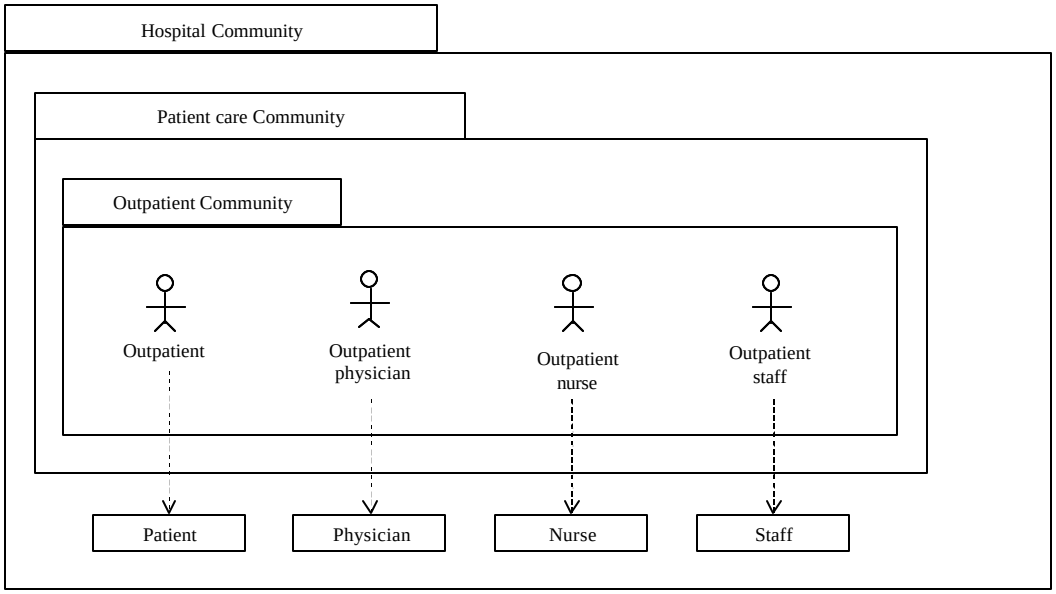


Figure 2: Outpatient Community

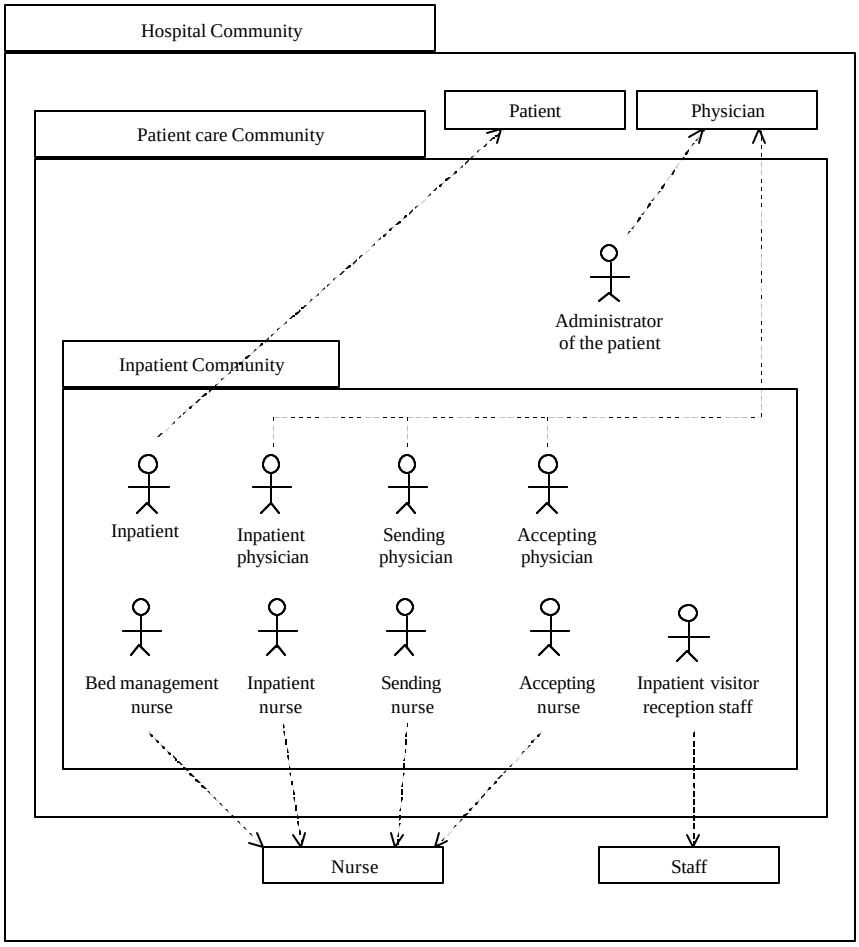


Figure 3: Inpatient Community

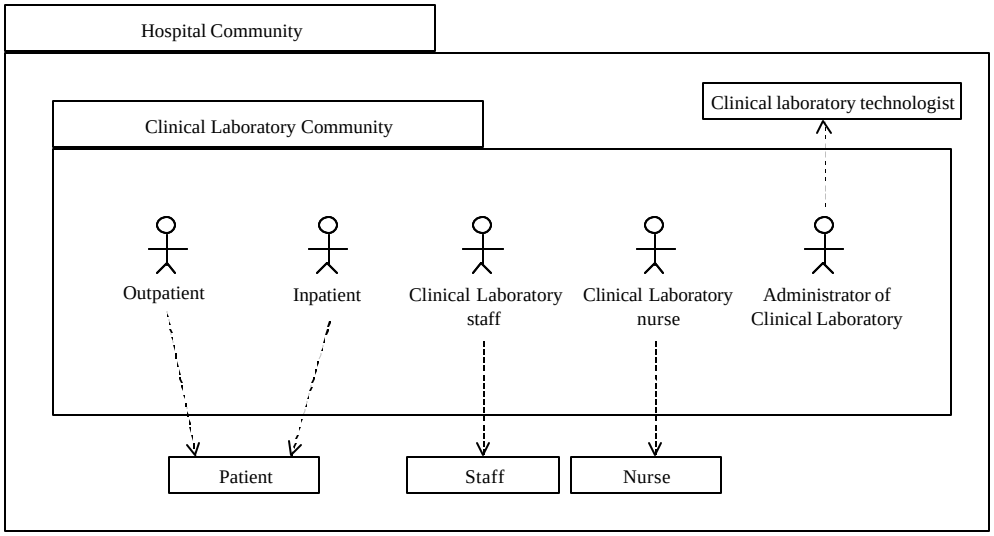


Figure 4: Clinical Laboratory Community

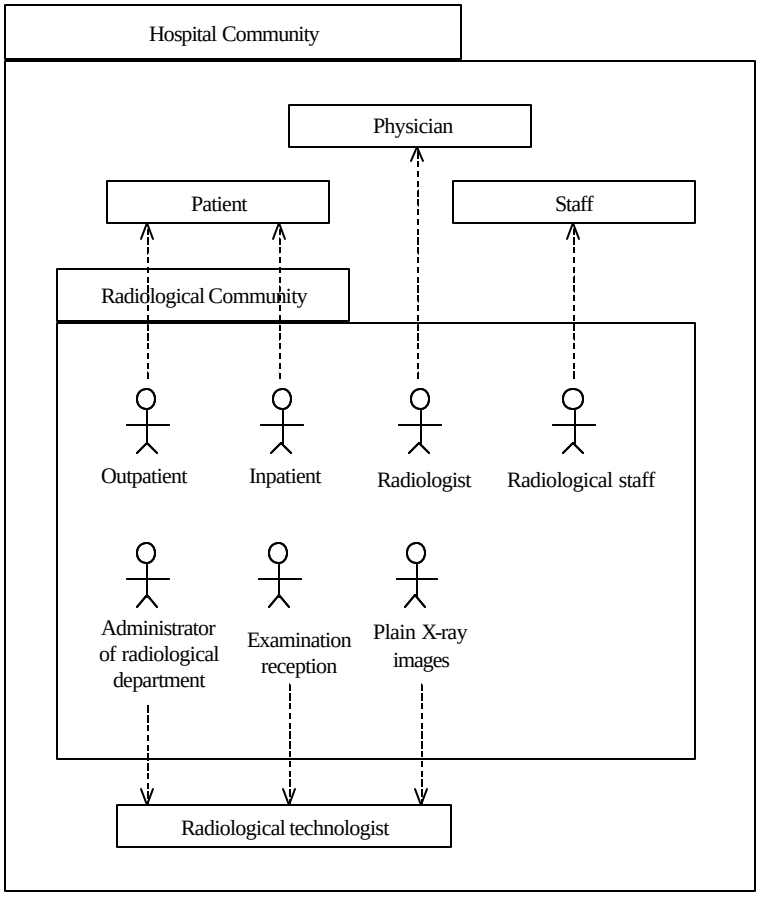


Figure 5: Radiological Community

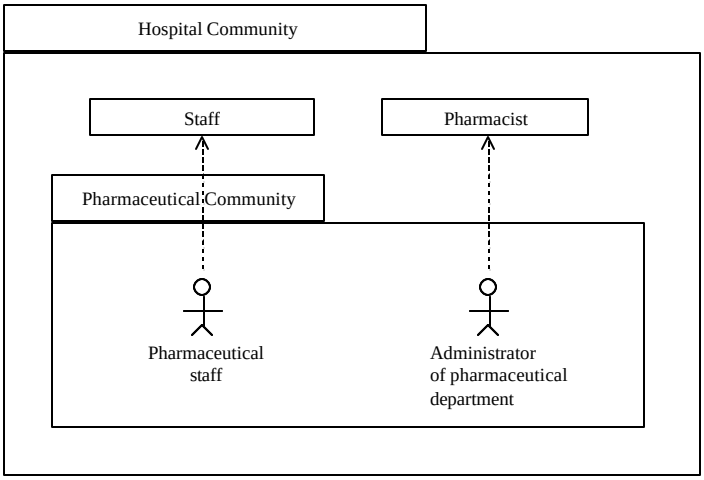


Figure 6: Pharmaceutical Community

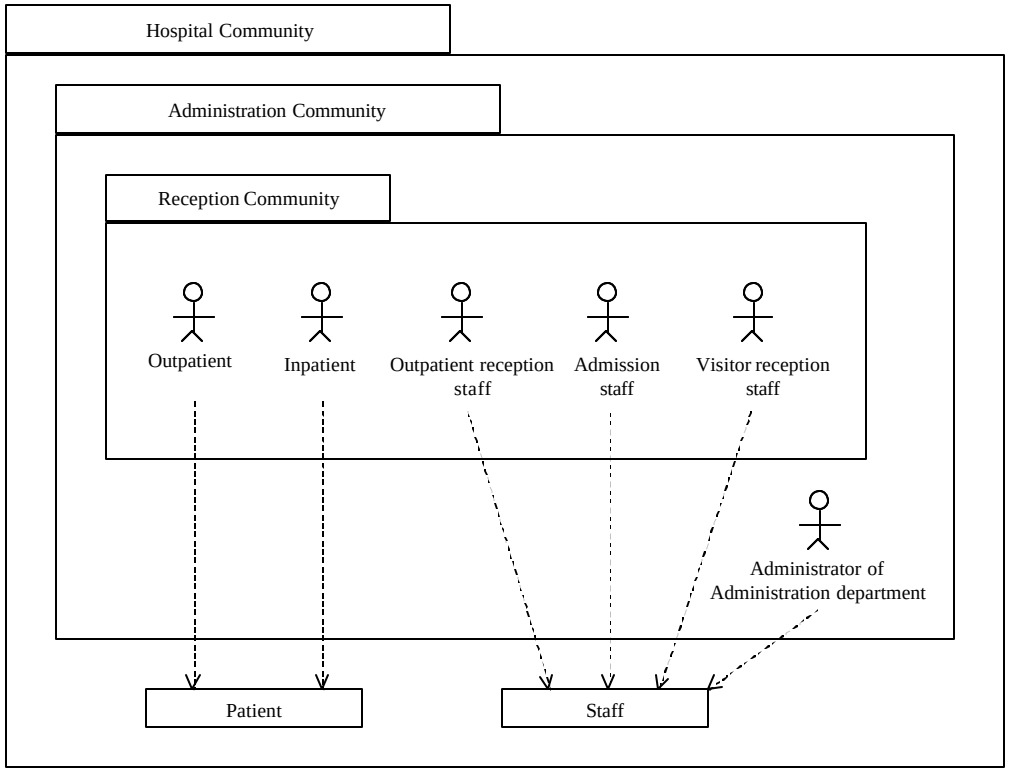


Figure 7: Reception Community

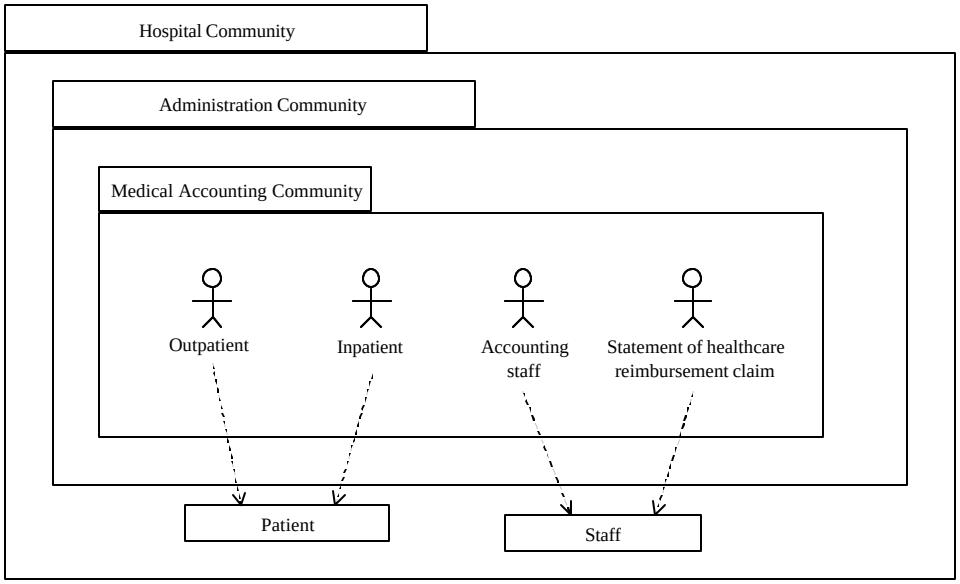


Figure 8: Medical Accounting Community

5.1.2 Objectives of each community

The objectives of each community are described below.

(1) Patient Care Community

- Providing patient care activities as a sub-community of the Hospital Community

(2) Outpatient Community

- Providing outpatient care as a sub-community of the Patient Care Community

(3) Inpatient Community

- Providing inpatient care as a sub-community of the Patient Care Community

(4) Clinical Laboratory Community

- Performing laboratory tests as a sub-community of the Hospital Community

(5) Radiological Community

- Performing X-ray examinations as a sub-community of the Hospital Community

(6) Pharmaceutical Community

- Auditing prescriptions issued by physicians and dispensing medicine according to prescriptions as a sub-community of the Hospital Community

(7) Administration Community

- Providing administrative services including reception and medical accounting as a sub-community of the Hospital Community

(8) Reception Community

- Performing clerical and reception activities as a sub-community of the Administration Community

(9) Medical Accounting Community

- Performing medical accounting activities as a sub-community of the Administration Community

5.2 Radiological Community

Among the communities listed in 5.1.1, this annex uses the Radiological Community to describe the details of specification using the UML Profile of EDOC. This section describes the Enterprise Viewpoint Specification for this community.

5.2.1 Scope

The scope of this community is as follows:

- Taking X-ray images
- Interpreting X-ray images
- Managing X-ray images

5.2.2 Enterprise objects

The following enterprise objects participate in this community and perform the roles described in .5.2.3.

- Patient
- Staff
- RadTechnologist (Radiological technologist)
- Physician
- HospitalInfoODPSystem (Hospital information ODP system)
- SystemAdmin (System administrator)

5.2.3 Roles

Detailed roles that are required for this community to function are listed below. Several roles have been refined with <<refine>>.

- Outpatient
- (Enterprise Object: Patient)
- Inpatient
- (Enterprise Object: Patient)
- RadStaff (Radiological staff)
- (Enterprise Object: Staff)
- ExamReception (Examination reception)
- (Enterprise Object: RadTechnologist)
- PlainX-rayImg (Plain X-ray images)
- (Enterprise Object: RadTechnologist)
- Radiologist
- (Enterprise Object: Physician)
- EmgExamOrder (Emergency examination order)
- (Enterprise Object: Physician)
- AdminRadDpt (Administrator of the radiological department)
- (Enterprise Object: SystemAdmin)
- PatientCertification
- GetExamOrder (Obtaining an examination order)
- GetPfmExamList (Obtaining a list of performed examinations)
- GetExamRslt (Obtaining an examination result)

<refined as:>

GetPfmExam (Obtaining a performed examination)
GetExamImg (Obtaining an examination images)

- GetPreReadInfo (Obtaining pre-reading information)

<refined as:>

GetInterpretImg (Obtaining images for interpretation)
GetPrevImg (Obtaining previous images)
GetExamOrder (Obtaining an examination order)
GetExamInfo (Obtaining examination information)
GetPatientRec (Obtaining a patient record)

- ArchExamRslt (Archiving an examination result)

<refined as:>

ArchExamImg (Archiving examination images)
ArchExamInfo (Archiving examination information)

- ArchInterpretReport (Archiving interpretation report)

<<refine>>

5.2.4 Policies

Here are some policies (constraints) placed on objects and roles.

(1) Administrator of the radiological department

- The administrator of the radiological department is obligated to record radiation exposure information and to archive the record for five years. (Archiving of medical records is mandated by the Medical Practitioners Law.)

- The administrator of the radiological department is obligated to archive the examination images for two years (as mandated by the Medical Radiological Technologists Law).

(2) Radiological technologist

- If the radiological technologist has any doubts regarding the contents of an examination order, the radiological technologist is obligated to resolve the doubts by submitting an inquiry to the physician who issued the order.

- The radiological technologist is obligated to perform radiological examination quickly without subjecting the patient to unreasonable discomfort or anxiety.

- The radiological technologist is obligated to understand the intent of the physician and to perform radiological examination in the manner that satisfies the physician's intent.

- The radiological technologist is obligated to perform radiological examination without delay according to the type of the physician's request (urgent, routine, etc.) and to report the examination results (including image delivery) to the physician.

- The radiological technologist is obligated to notify the Medical Accounting Community of the accounting information related to the examination process without delay.

- The radiological technologist is obligated to maintain the equipment used for radiological examination so that the equipment meets its performance specifications. The radiological technologist is also obligated to appropriately replenish and manage materials required to perform radiological examination.

- The radiological technologist is obligated to protect all of the patient's personal information obtained in the process of the technologist's activities.

(3) Radiologist

- The radiologist is obligated to submit the interpretation report to the physician without delay according to the type of the physician's request (degree of urgency, reason for radiography, etc.).

(4) Physician

- The physician who orderes the radiological examination is obligated to sign the record of the radiation dose.

(5) Others

- Only physicians and radiological technologists are permitted to expose a human body to radiation (as mandated by the Medical Radiological Technologists Law).

- The hospital information ODP system is obligated to electronically archive the medical images used for diagnosis as authorized images. Images acquired but not used

for diagnosis need not be archived. (It is assumed that the hospital information ODP system includes an image management system.)

5.2.5 Business Processes

This section describes three Business processes in the Radiological Community: **same day radiological examination reception**, **plain X-ray images**, and **interpretation**.

Sub-processes corresponding to activities such as **emergency cancellation of examination**, **instruct to move to other examinations**, and **assignment of examination images**, are also specified in the process **same day radiological examination reception**.

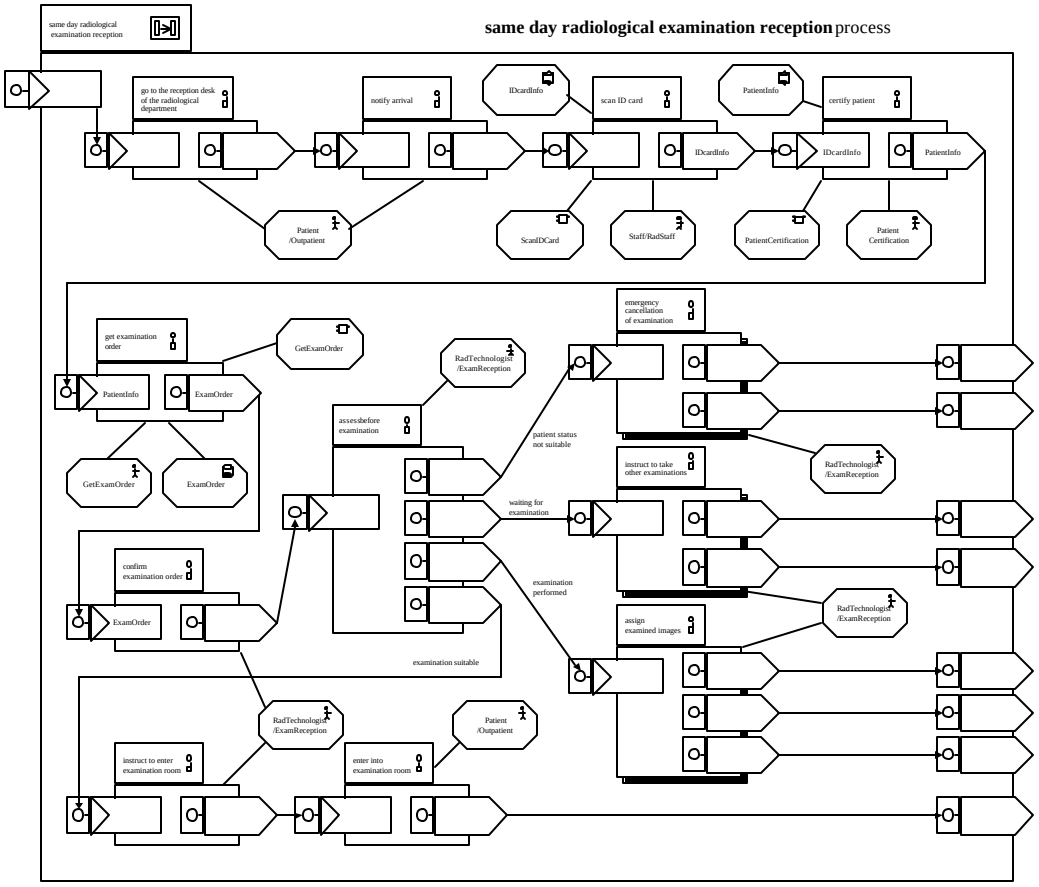


Figure 9: same day radiological examination reception (process)

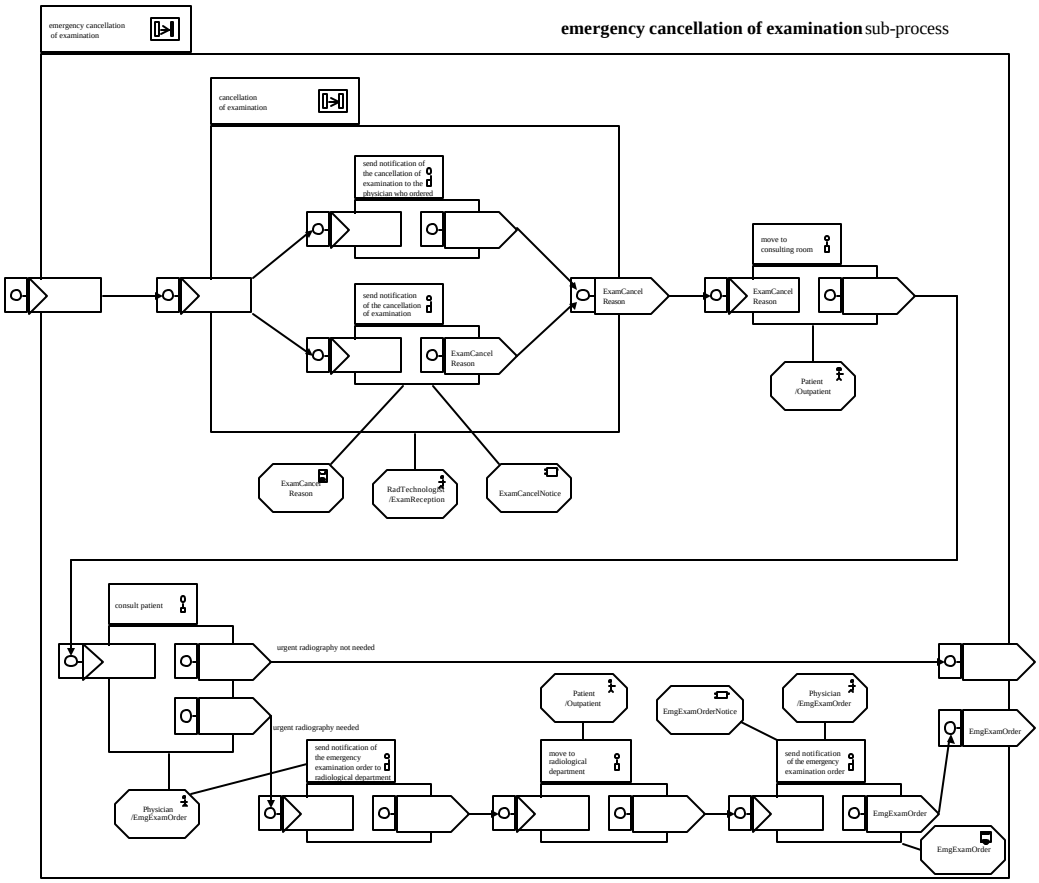


Figure 10: emergency cancellation of examination (sub-process)

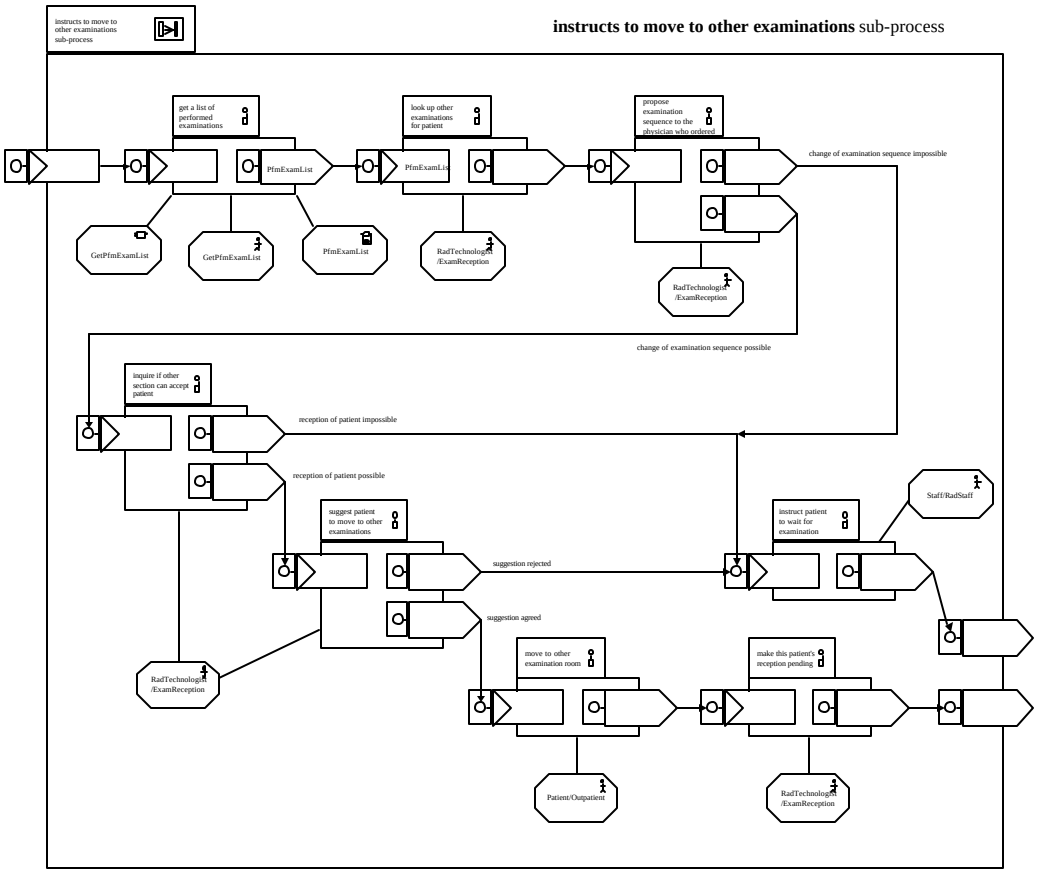


Figure 11: instruct to move to other examinations (sub-process)

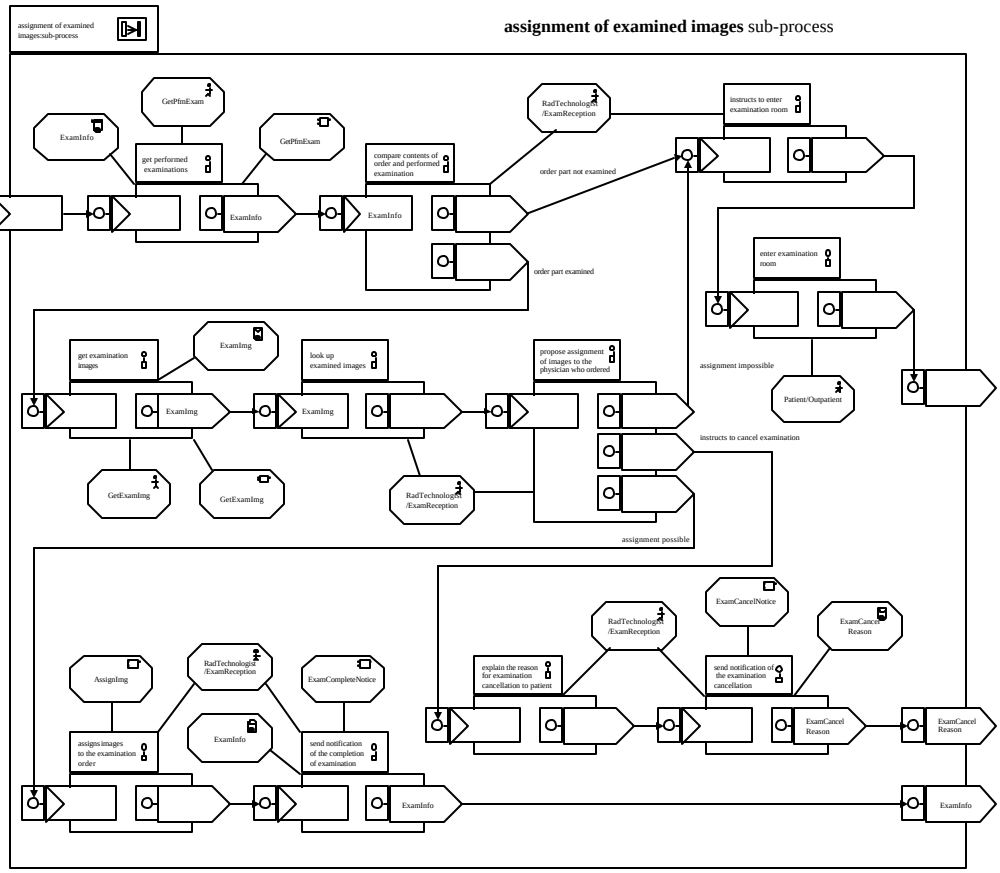


Figure 12: assignment of examined images (sub-process)

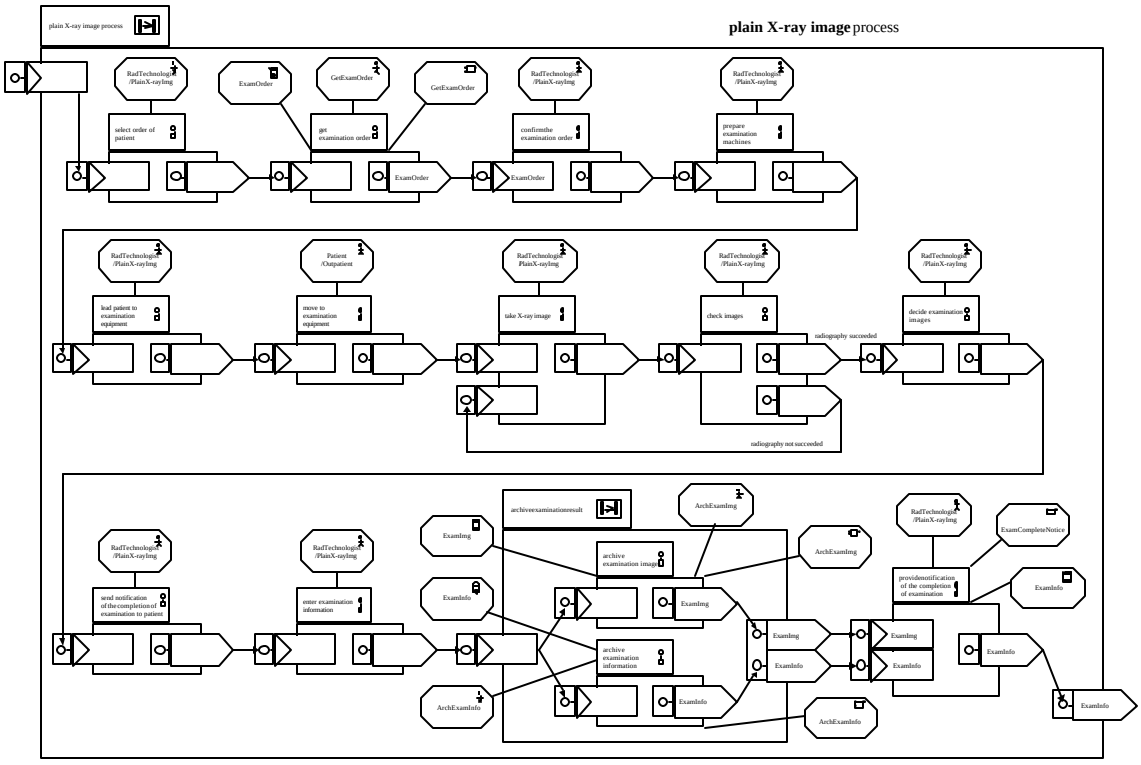


Figure 13: plain X-ray image (process)

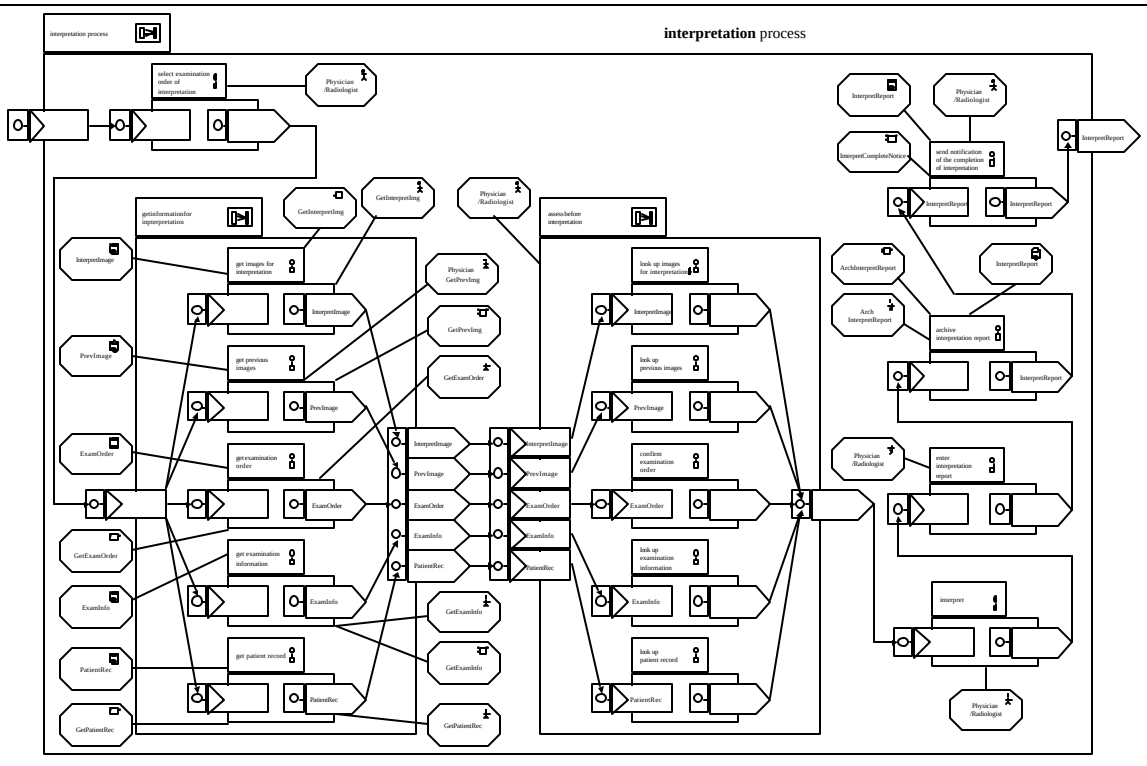


Figure 14: interpretation (process)

6. *Information Viewpoint*

The following figures represent the Information Viewpoint Specification for this community specified with the Entities Profile.

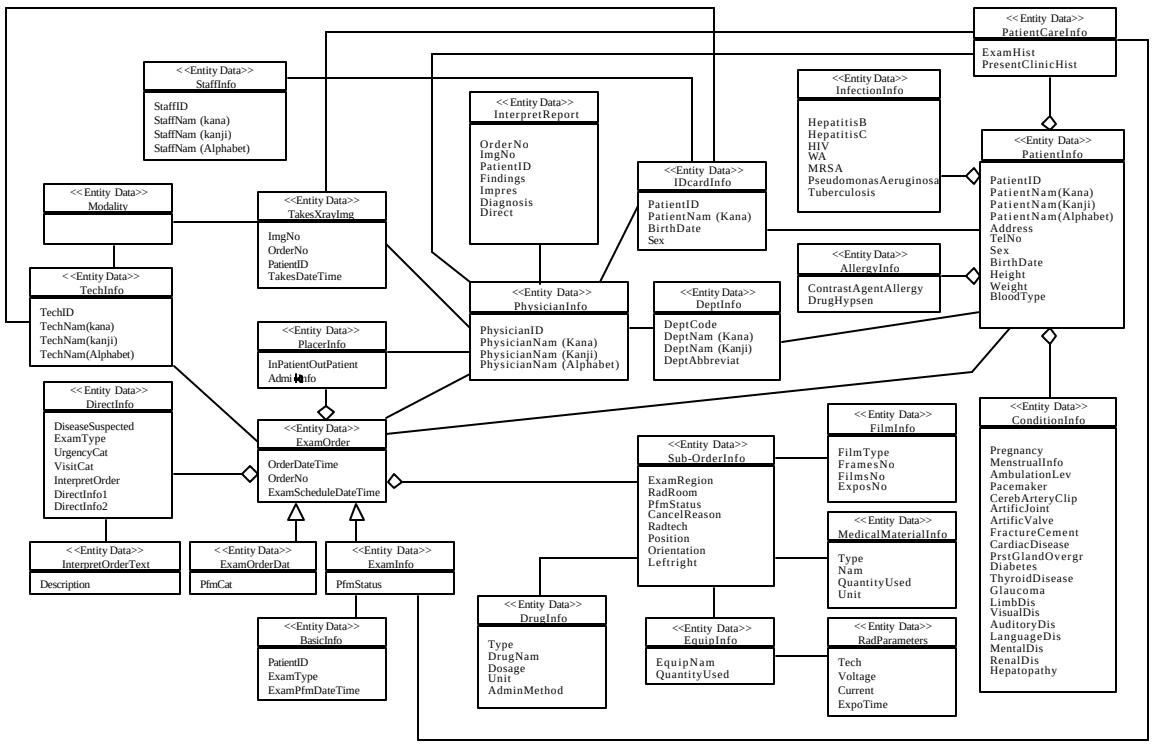


Figure 15: Information model (Information Viewpoint)

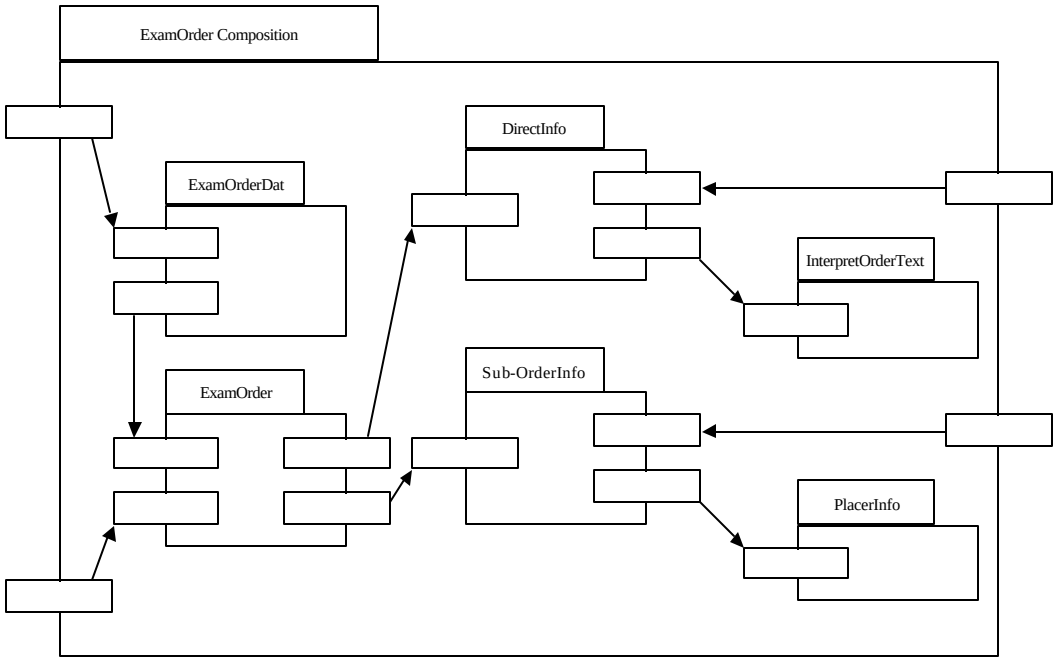


Figure 16: ExamOrder Composition (Composition Viewpoint)

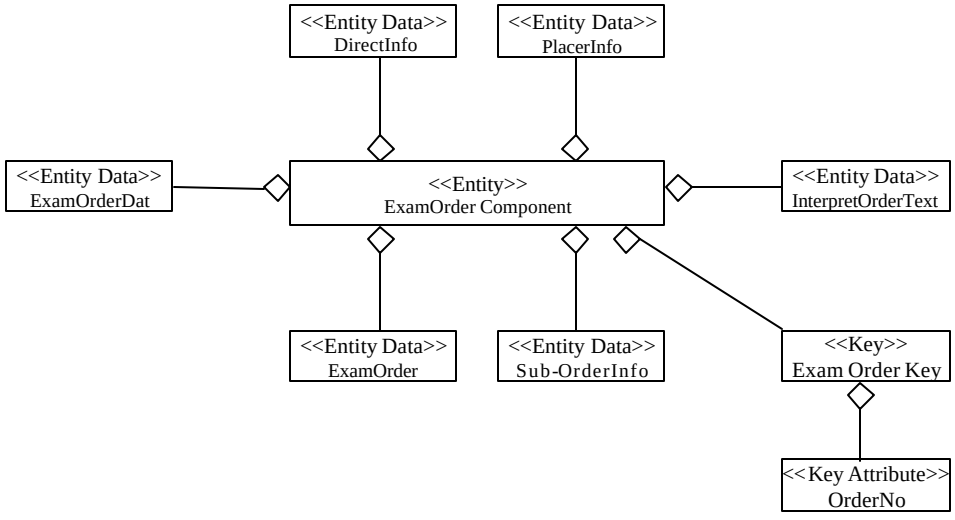


Figure 17: ExamOrder Component (Entity Viewpoint)

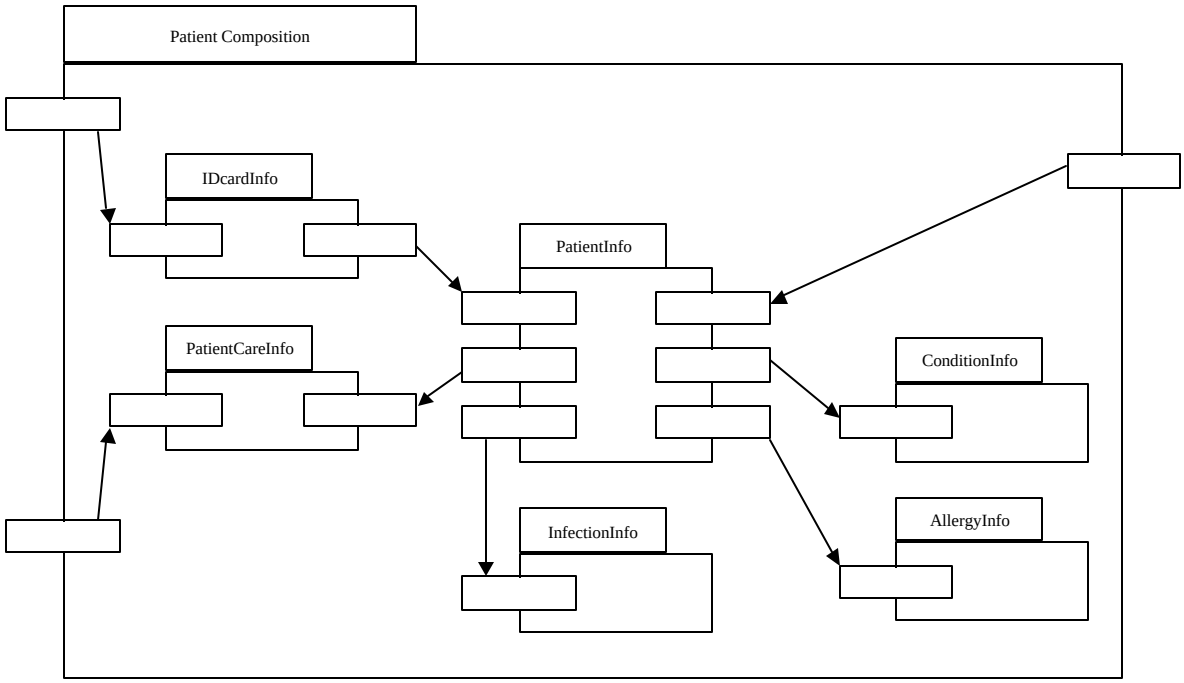


Figure 18: Patient Composition (Composition Viewpoint)

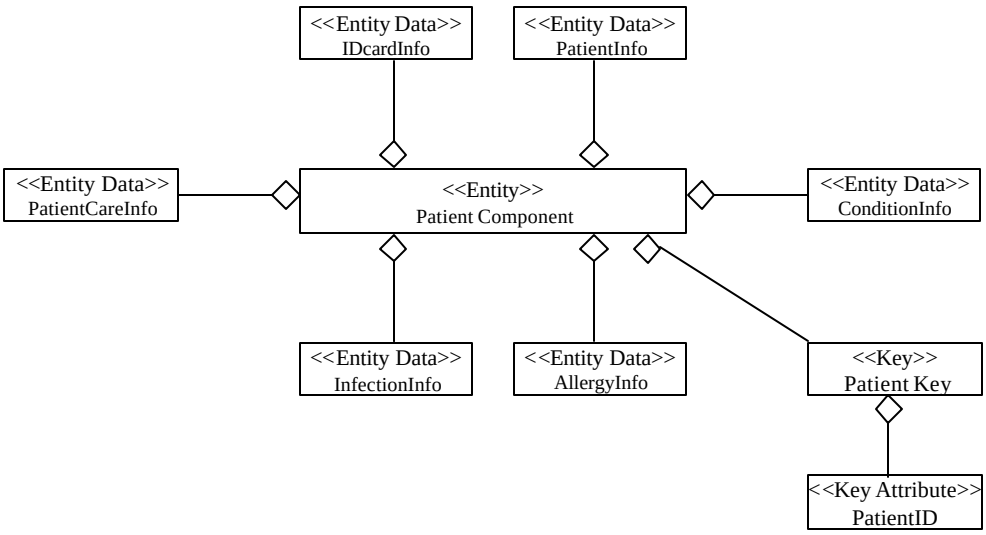


Figure 19: Patient Component (Entity Viewpoint)

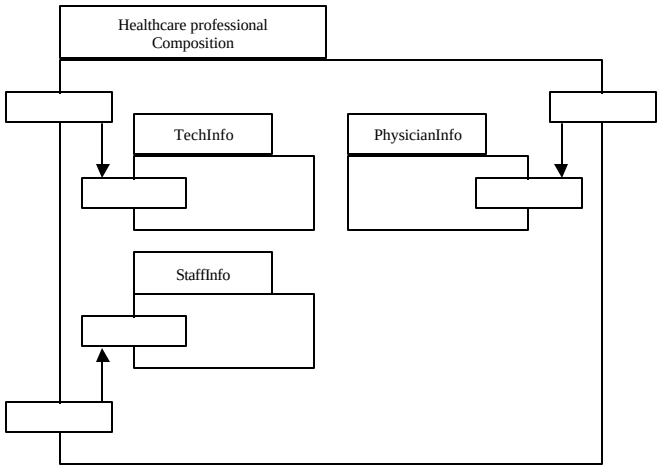


Figure 20: Healthcare professional Composition (Composition Viewpoint)

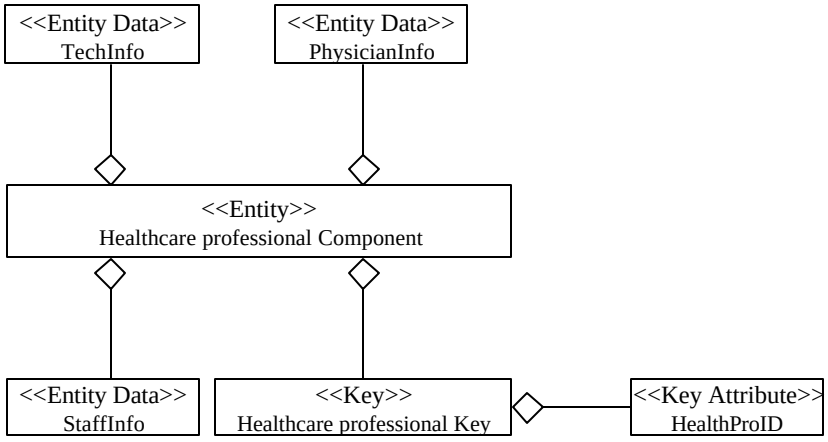


Figure 21: Healthcare professional Component (Entity Viewpoint)

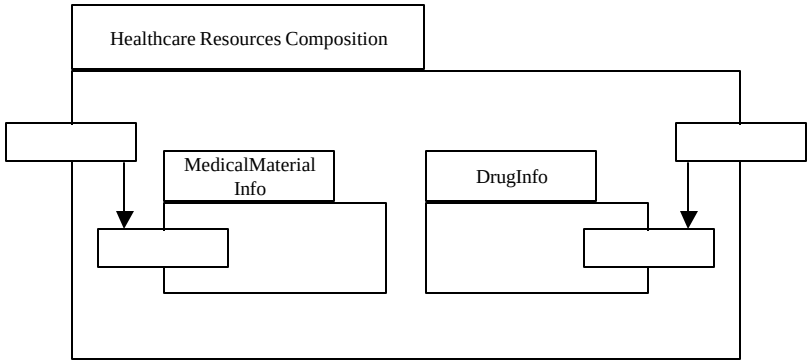


Figure 22: Healthcare Resource Composition (Composition Viewpoint)

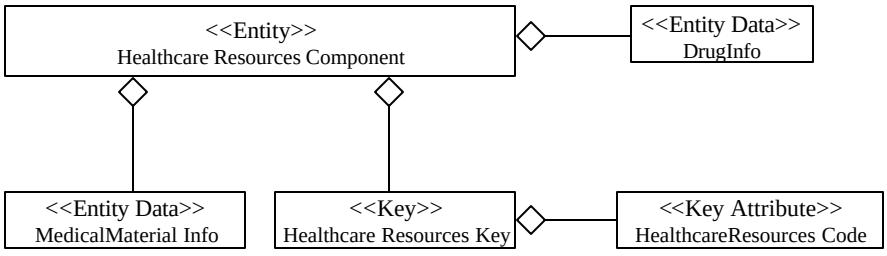


Figure 23: Healthcare Resource Component (Entity Viewpoint)

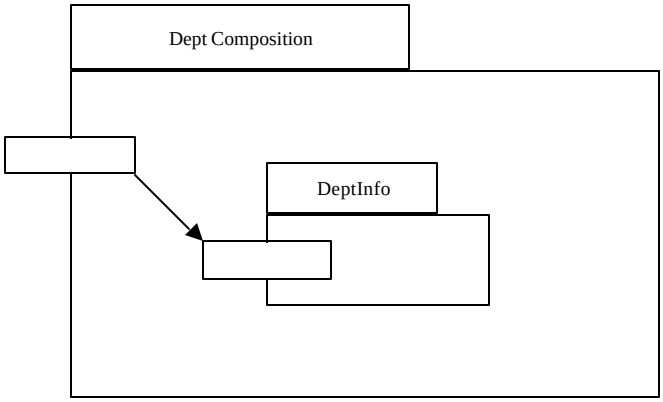


Figure 24: Dept Composition (Composition Viewpoint)

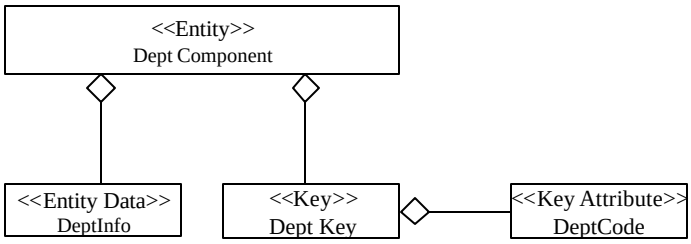


Figure 25: Dept Component (Entity Viewpoint)

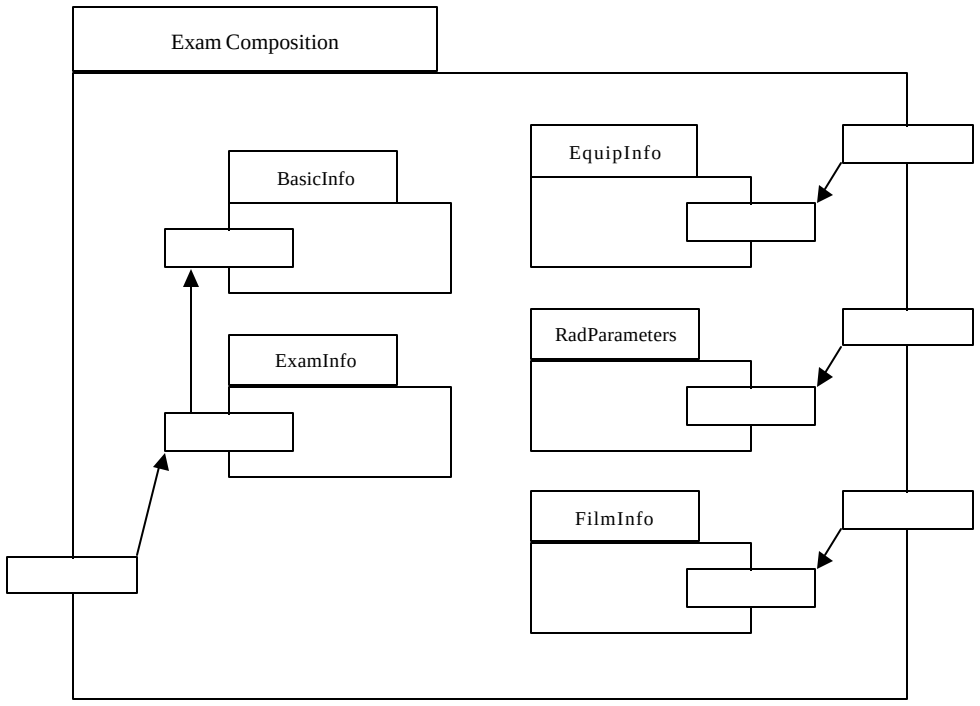


Figure 26: Exam Composition (Composition Viewpoint)

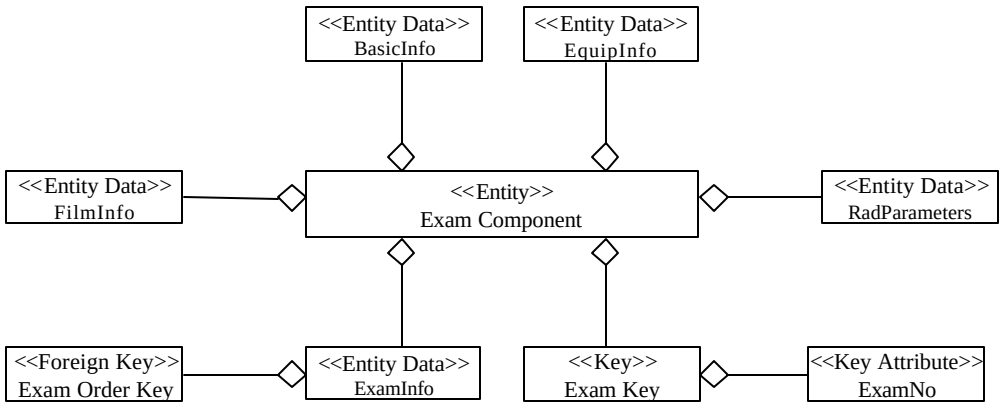


Figure 27: Exam Component (Entity Viewpoint)

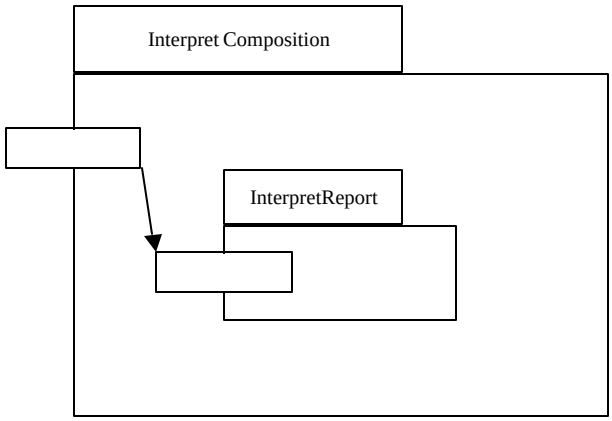


Figure 28: Interpret Composition (Composition Viewpoint)

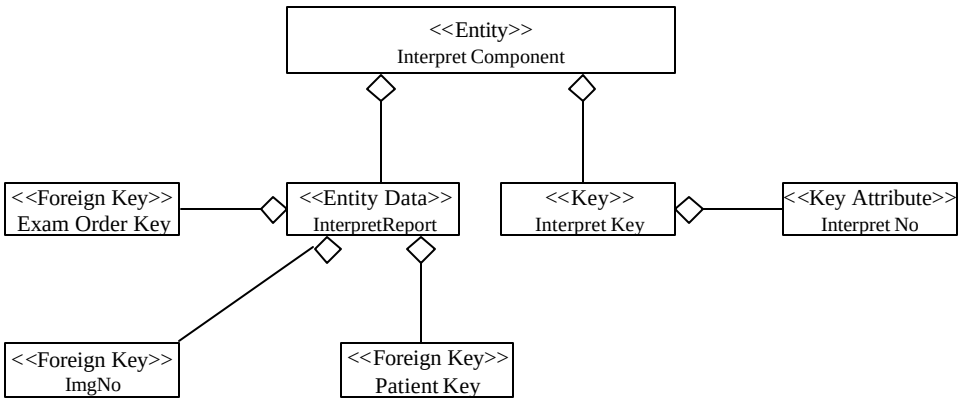


Figure 29: Interpret Component (Entity Viewpoint)

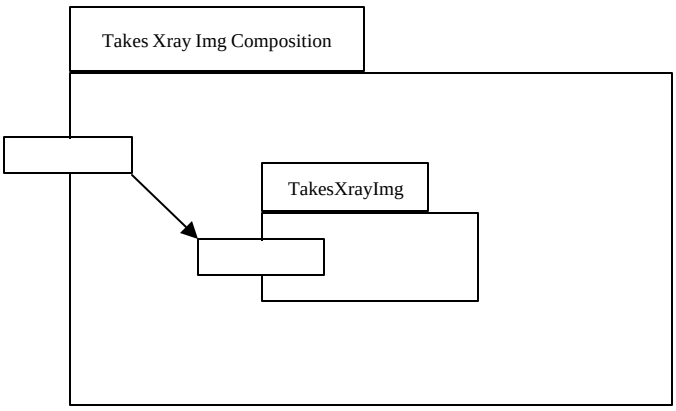


Figure 30: Takes Xray Img Composition (Composition Viewpoint)

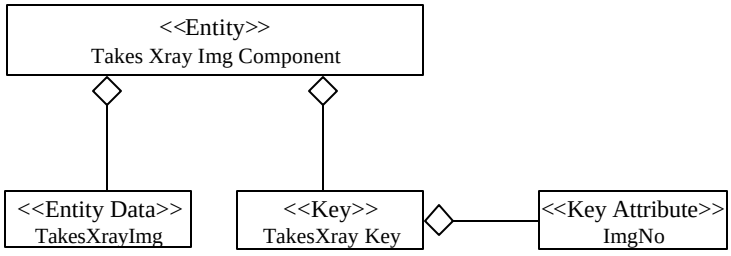


Figure 31: Takes Xray Img Component (Entity Viewpoint)

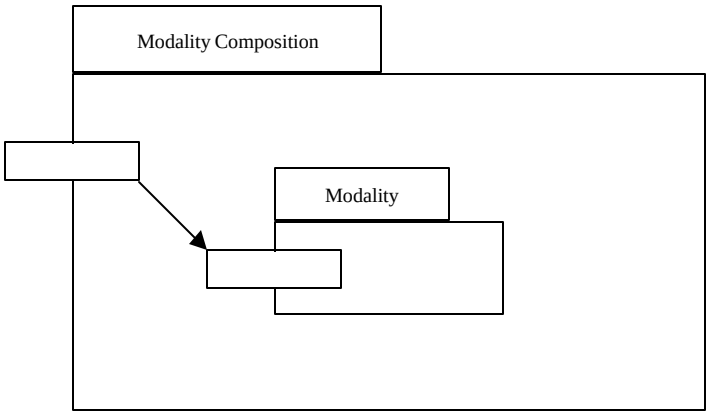


Figure 32: Modality Composition (Composition Viewpoint)

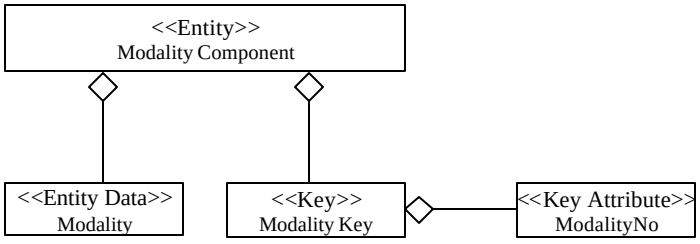


Figure 33: Modality Component (Entity Viewpoint)

7. Computational Viewpoint Specification

In the Computational Viewpoint Specification, computational objects are derived and presented as CCA ProcessComponents, followed by PORT specifications as interface specifications for computational objects. And, Protocol specifications as interaction specifications between computational objects are described. Component Collaboration Architecture Profile (Part III a) is the main Profile used.

7.1 Identified set of ProcessComponents (computational objects)

From the Enterprise Viewpoint Specification and Information Viewpoint Specification, the following components are identified as computational objects:

- ScanIDCard (Component for scanning ID Card)
- PatientCertification (Component for patient certification)
- GetExamOrder (Component for obtaining examination order)
- ExamCancellNotice (Component for examination cancellation notice)
- EmgExamOrderNotice (Component for emergency examination order notice)
- GetPfmExamList (Component for obtaining a list of performed examinations)
- GetPfmExam (Component for obtaining performed examinations)
- GetExamImg (Component for obtaining examination images)
- ArchExamImg (Component for examination images archiving)
- AssignImg (Component for assign image)
- ArchExamInfo (Component for examination information archiving)
- ExamCompleteNotice (Component for completion of examination notice)
- GetInterpretImg (Component for obtaining images for interpretation)
- GetPrevImg (Component for obtaining previous images)
- GetExamInfo (Component for obtaining examination information)
- GetPatientRec (Component for obtaining patient records)
- ArchInterpretReport (Component for interpretation report archiving)
- InterpretCompleteNotice (Component for completion of interpretation notice)

7.2 PORT (interface) specification

The following figures represent the PORT specifications for each ProcessComponent using CCA.

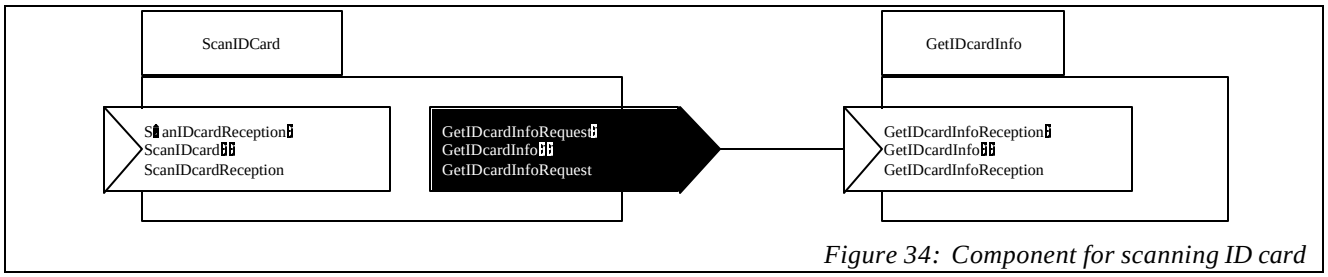


Figure 34: Component for scanning ID card

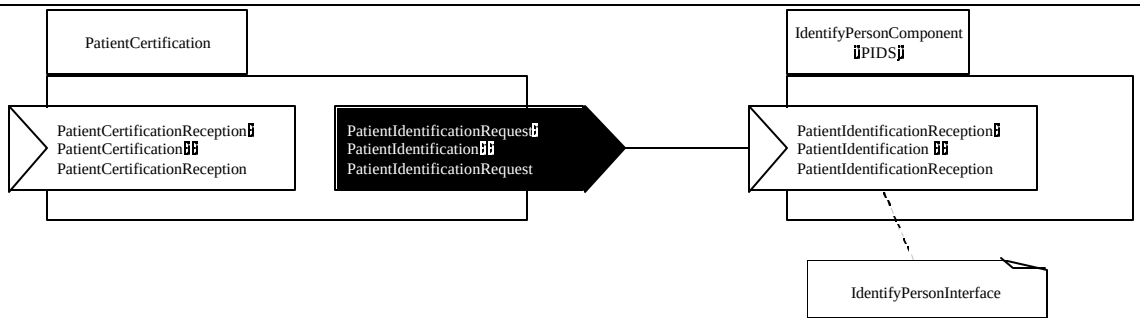


Figure 35: Component for patient certification

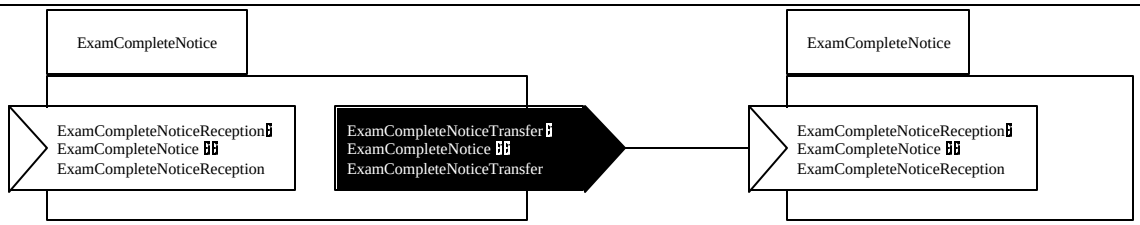


Figure 36: Component for obtaining an examination order

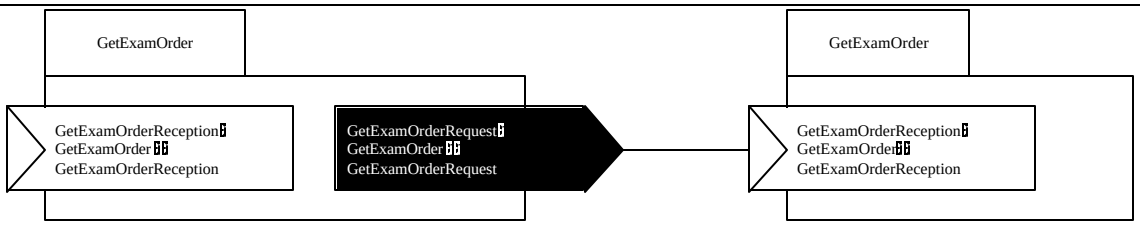


Figure 37: Component for completion of an examination notice

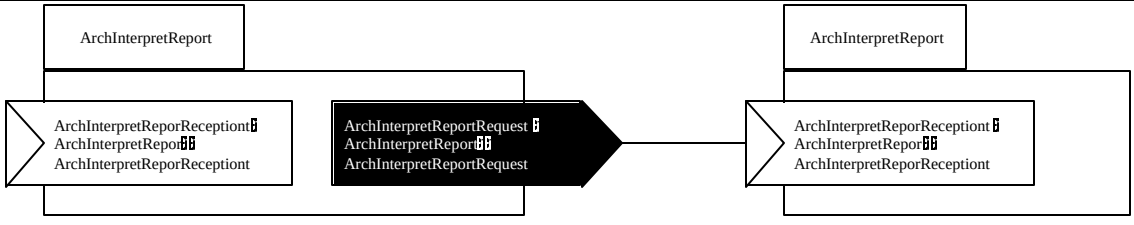


Figure 38: Component for archiving an interpretation report

Note that the component for patient certification uses the IdentifyPersonComponent in the PIDS (Person Identification Service) for the identification of the patients (reference: OMG Healthcare DTF: Document Number: corbamed/98-02-29: Final adopted PIDS specification including errata sheets). In such cases as obtaining or archiving information, get/archive messages are sent to the system that stores the information.

7.3 Protocol (interaction) specification

The following figures represent protocol specifications for ProcessComponents (roles) using CCA.

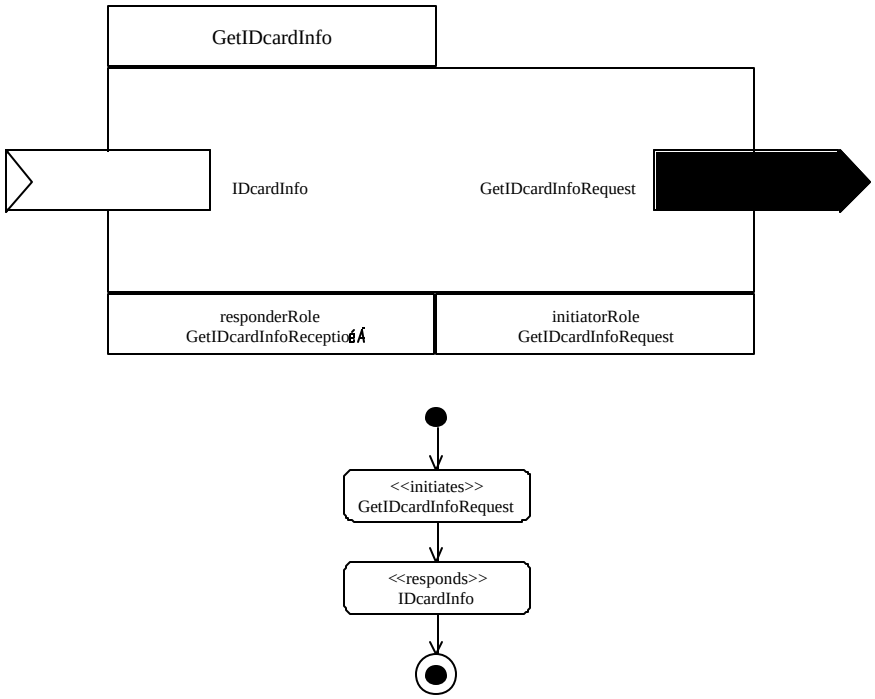


Figure 39: Protocol for getting ID card information

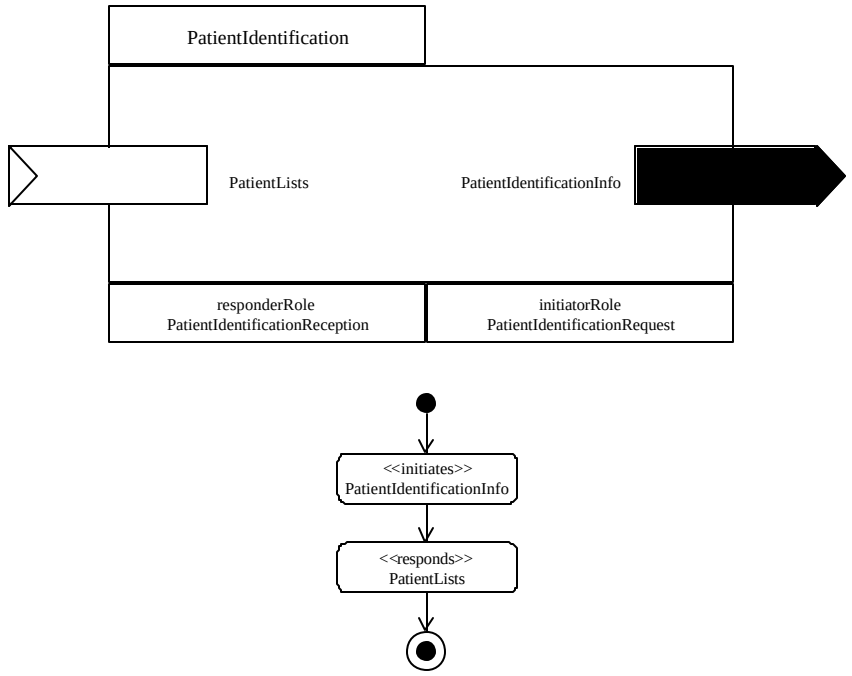


Figure 40: Protocol for patient identification

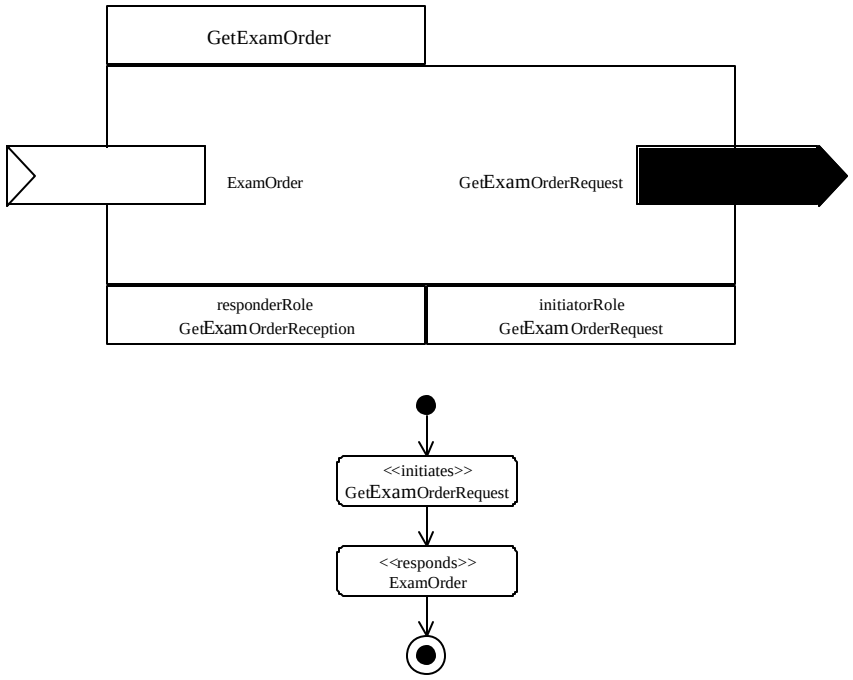


Figure 41: Protocol for getting examination order

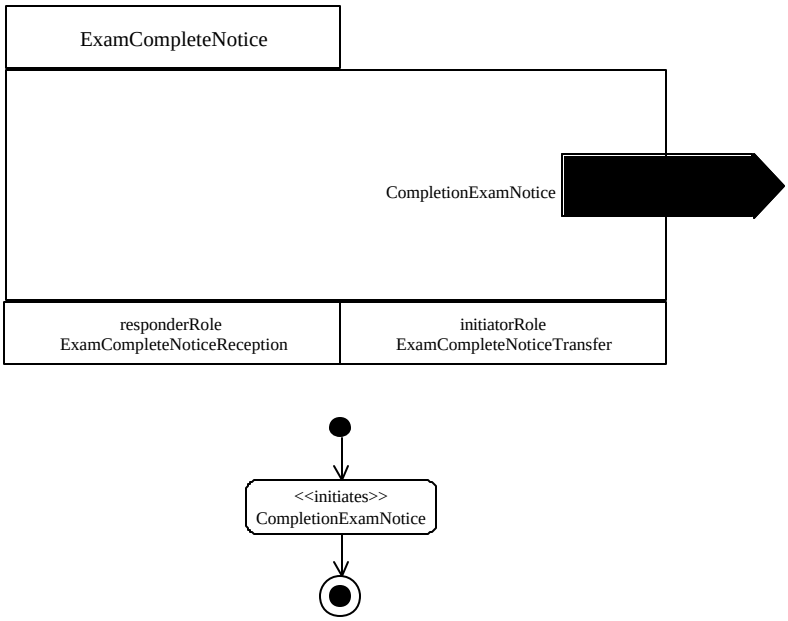


Figure 42: Protocol for completion of examination notice

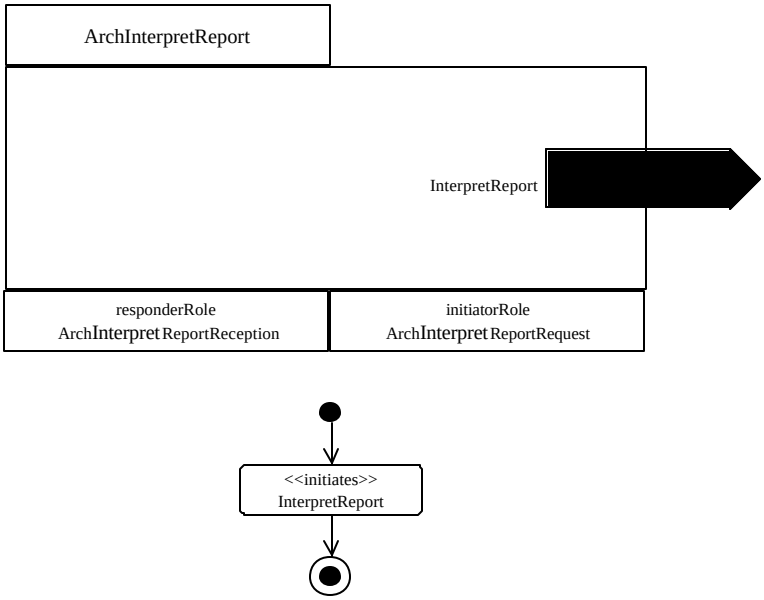


Figure 43: Protocol for archiving interpretation report

Annex D - Examples of Patterns

Contents

List of Figures

1. Simple Pattern Examples

Figure 1 below represents the business process pattern in ECA. An order process pattern is applied to a customer/ supplier model.

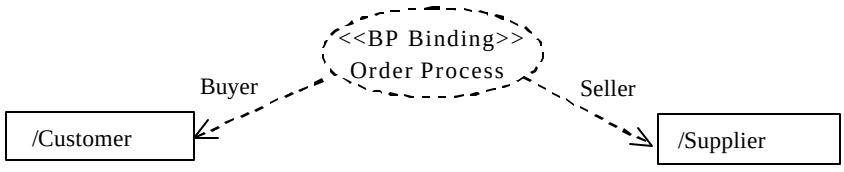


Figure 1 Parameterized Collaboration

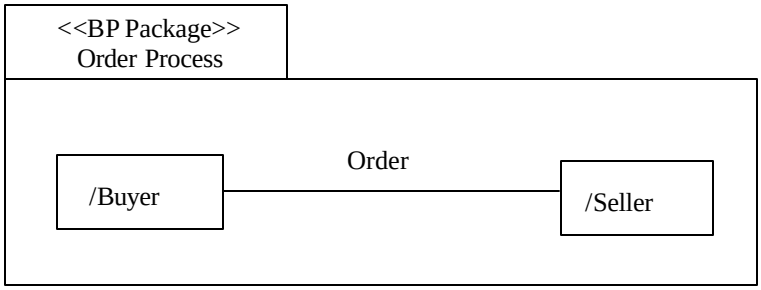


Figure 2 Order Process Pattern

Figure 2 represents an order process pattern. It defines a relationship between a buyer and a seller in the order process using a pattern to make it reusable model.

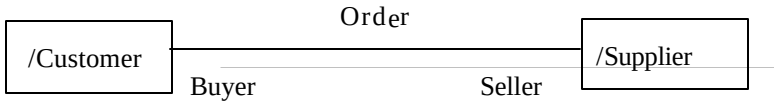


Figure 3 Customer/ Supplier Model

Figure 3 represents the unfolded order process pattern. An association between a customer and a supplier is expressed as role, a buyer and a seller, in the business process modeling.

Figure 4 and Figure 5 are the funds transfer pattern and the exchange pattern. Each one is a simple pattern, but it's possible to apply more than one pattern as Figure 6.

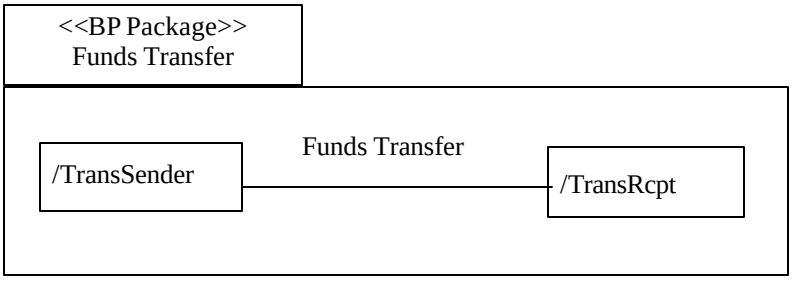


Figure 4 Funds Transfer Pattern

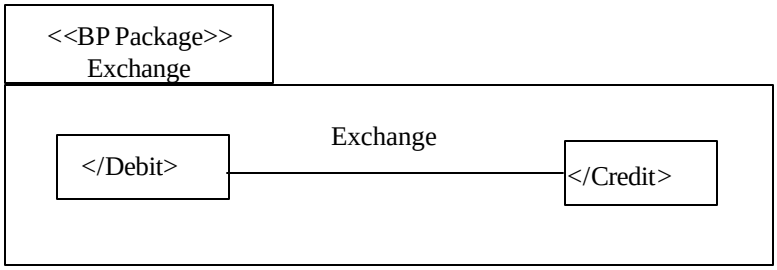


Figure 5 Exchange Pattern

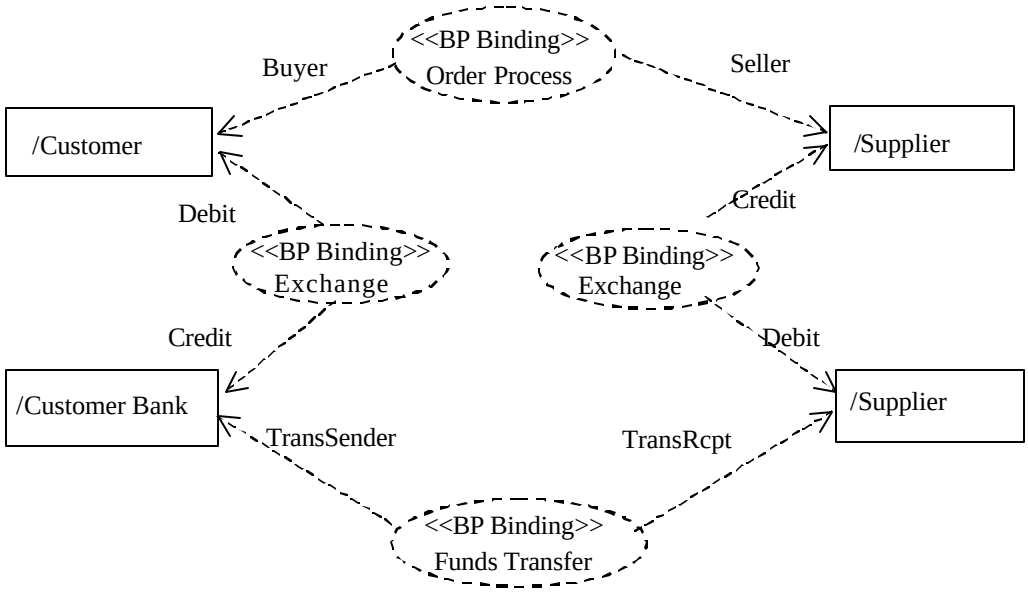


Figure 6 Purchase Model

Figure 6 represents collaboration by applying multiple patterns into the model. There is an order process pattern between a customer and a supplier, and an exchange process pattern is also applied between a customer and its bank to express the process of payment from a customer to its bank. Furthermore, the funds transfer pattern is applied between customer bank and supplier bank to process the transaction between two. At last, a supplier withdraws the payment from supplier bank. Figure 7 below represents the object model after all patterns are unfolded.

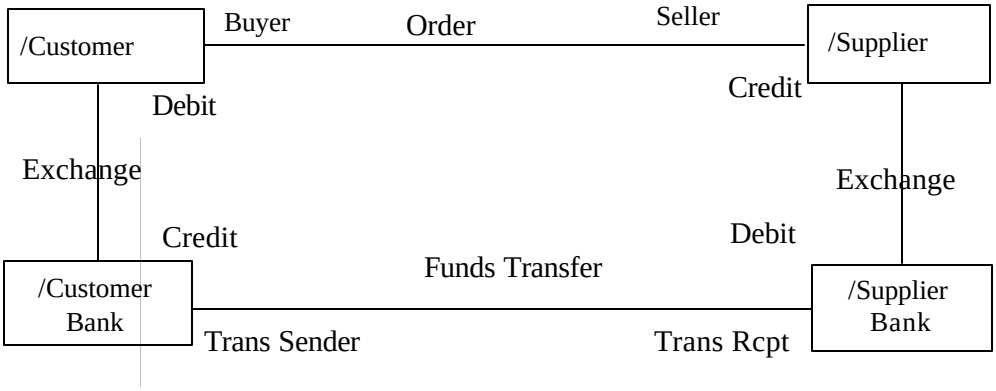


Figure 7 Unfolded Purchase Model

This example uses the patterns in one layer, but by applying patterns in several layers more effective and reusable model models can be created.

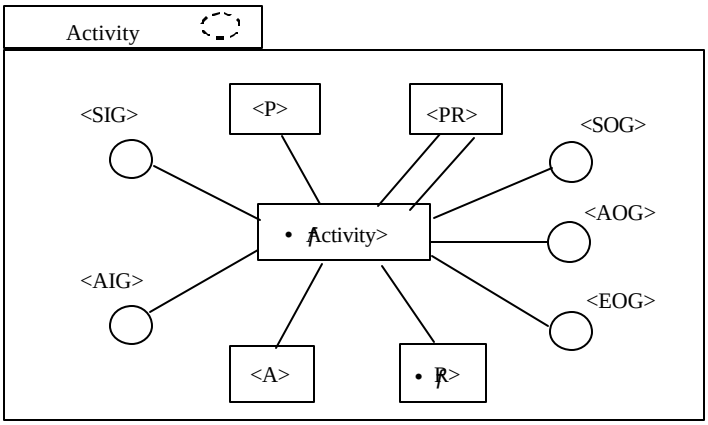
2. *Process Model Patterns*

Section 5.5 of Part I titled Process Model Patterns describes various patterns of common usage and associated special notion that may be useful when using the ECA Process Model. In there, the pattern in terms of its normal notation possibly with parameterized parts are described.

However, It is not sufficient to express the complexity required by these patterns, since they usually consist of a CompoundTask parameterized by an Activity that will have some unknown number of ProcessMultiPorts and ProcessFlowPorts. When instantiating such a template with respect to a particular Activity, the Compound Task needs to have corresponding ProcessMultiPorts and ProcessFlow Ports connected by Flows to the equivalent ports on the Activity argument to the template.

We show examples of describing the Process Model Patterns with the BP Package notation

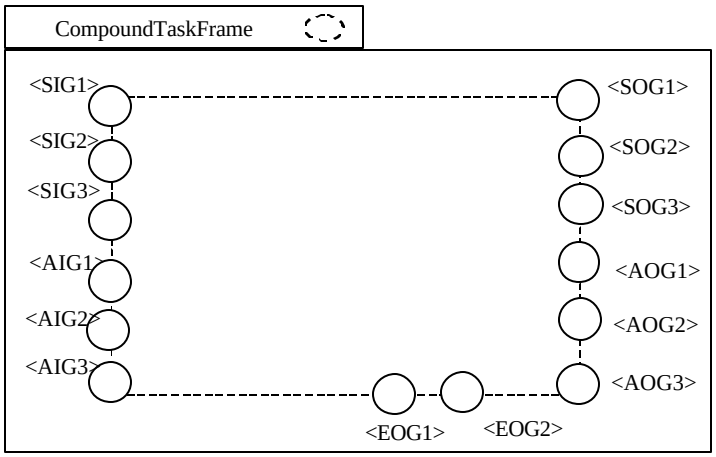
2.1 Activity



Note:
<SIG>: Sync InputGroup, <AIG>: Async InputGroup
<SOG>: Sync OutputGroup, <AOG>: Async OutputGroup
<EOG>: ExceptionGroup
<P>: Performer, <PR>:ProcessRole
<A>: Artifact, <R>: ResponsibleParty

Figure 8 Activity Pattern

2.2 CompoundTask



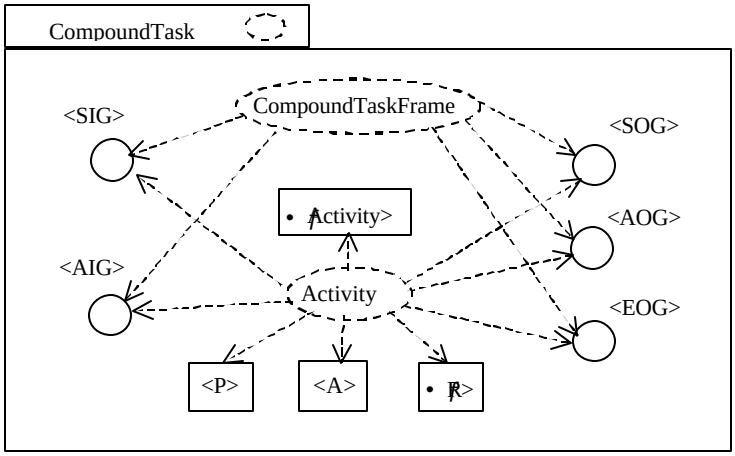


Figure 9 CompoundTaskFrame and CompoundTask pattern

2.3 ActivityPreCondition and ActivityPostCondition

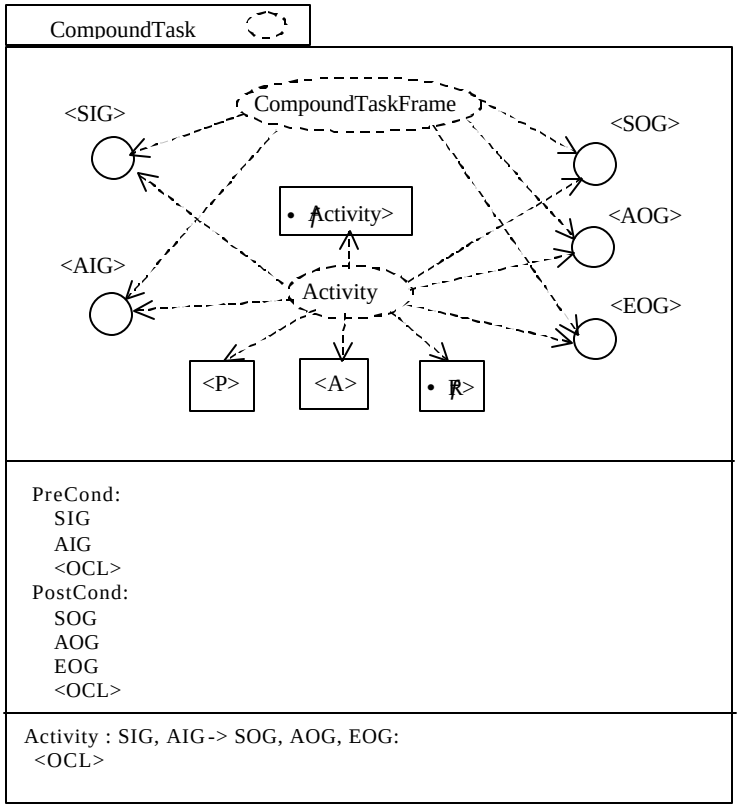
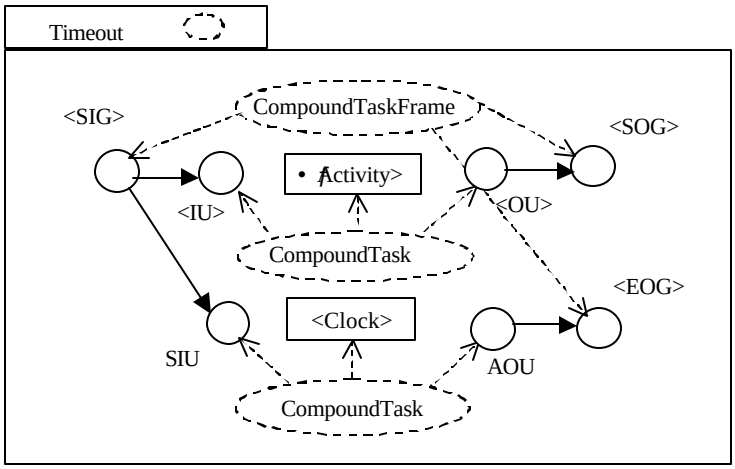


Figure 10 ActivityPreCondition and ActivityPostCondition Pattern

2.4 Timeout



<IU>: InputUsage
<OU>: OutputUsage

Figure 11 Timeout Pattern

2.5 Terminate

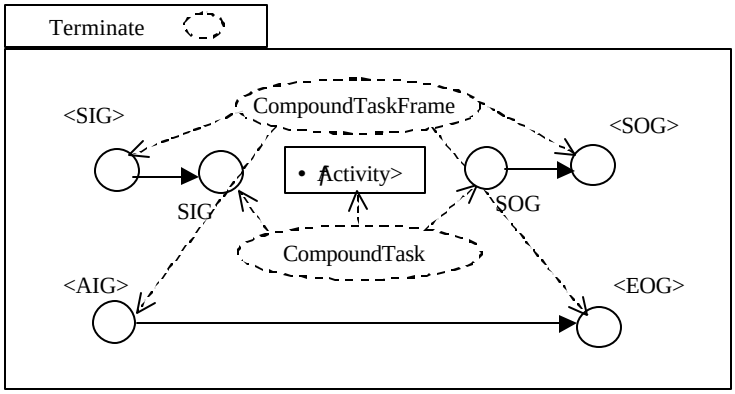


Figure 12 Terminate Pattern

2.6 Simple Loop

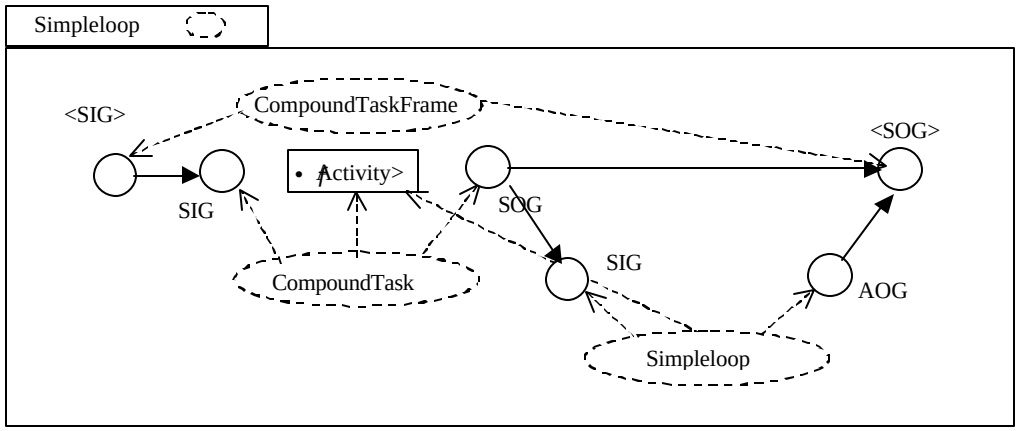


Figure 13 Simple Loop Pattern

2.7 While and Repeat/Until Loop

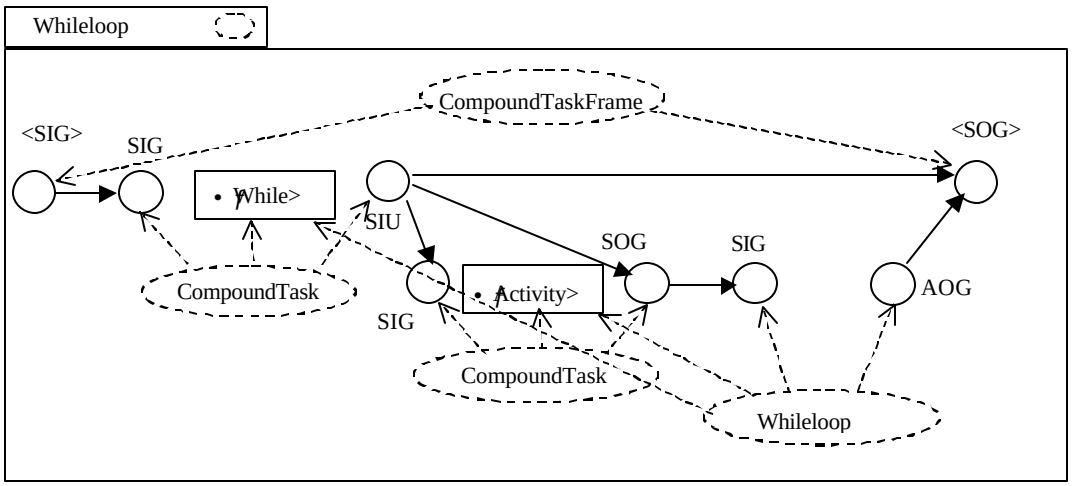


Figure 14 While Loop Pattern

2.8 For Loop

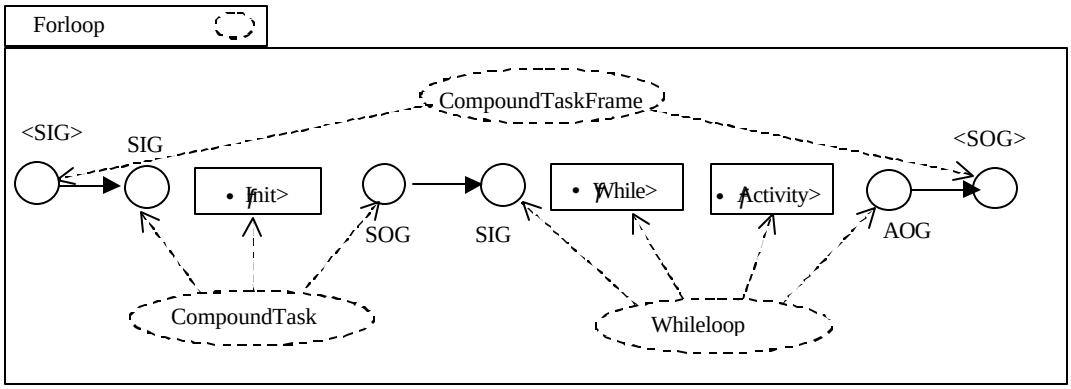


Figure 15 For Loop Pattern

2.9 Multitask

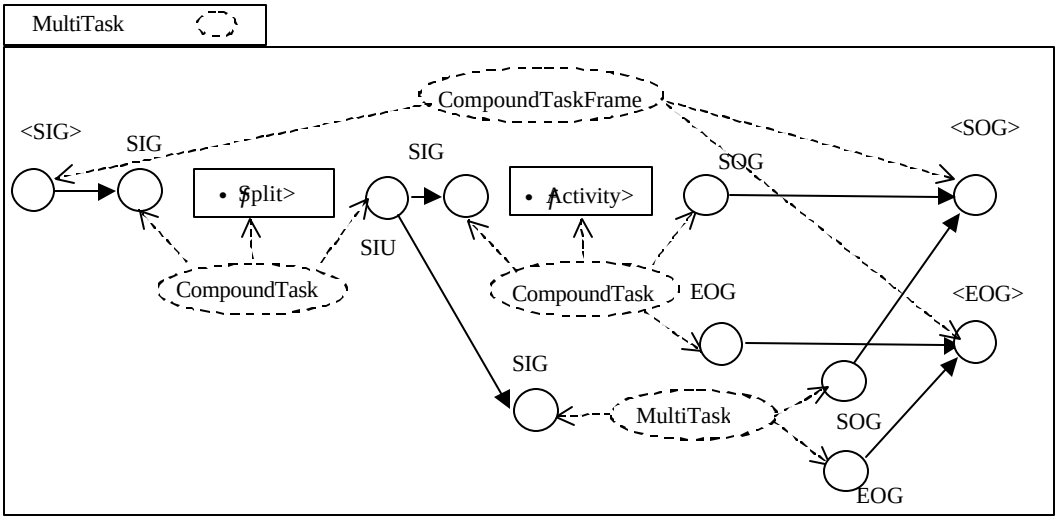


Figure 16 Multitask Pattern

2.10 Procurement

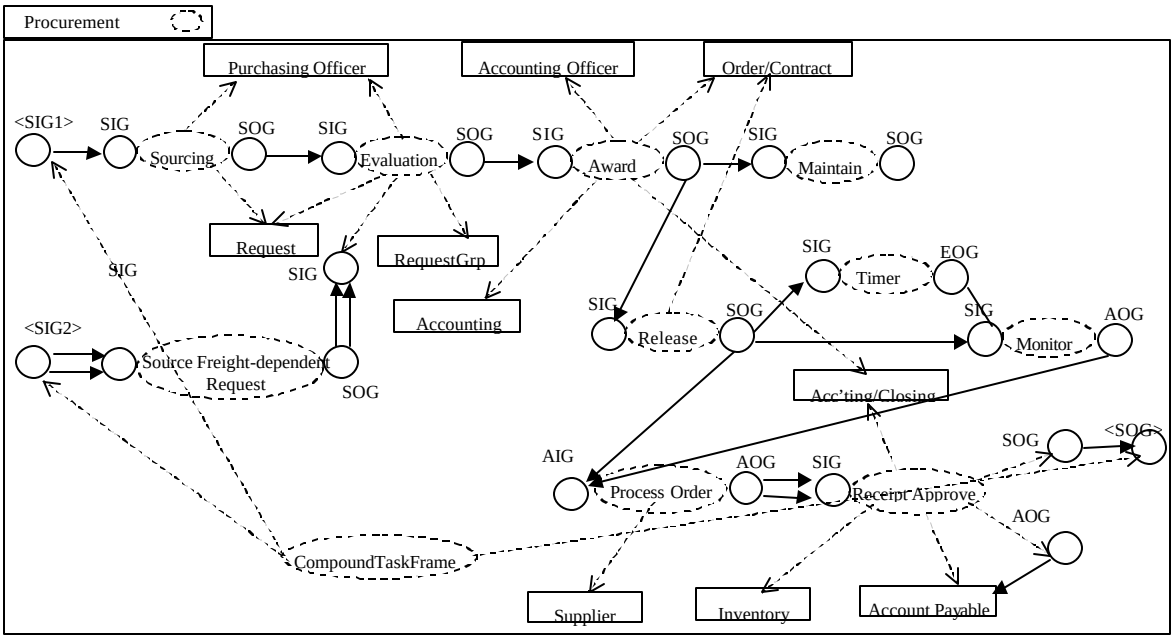
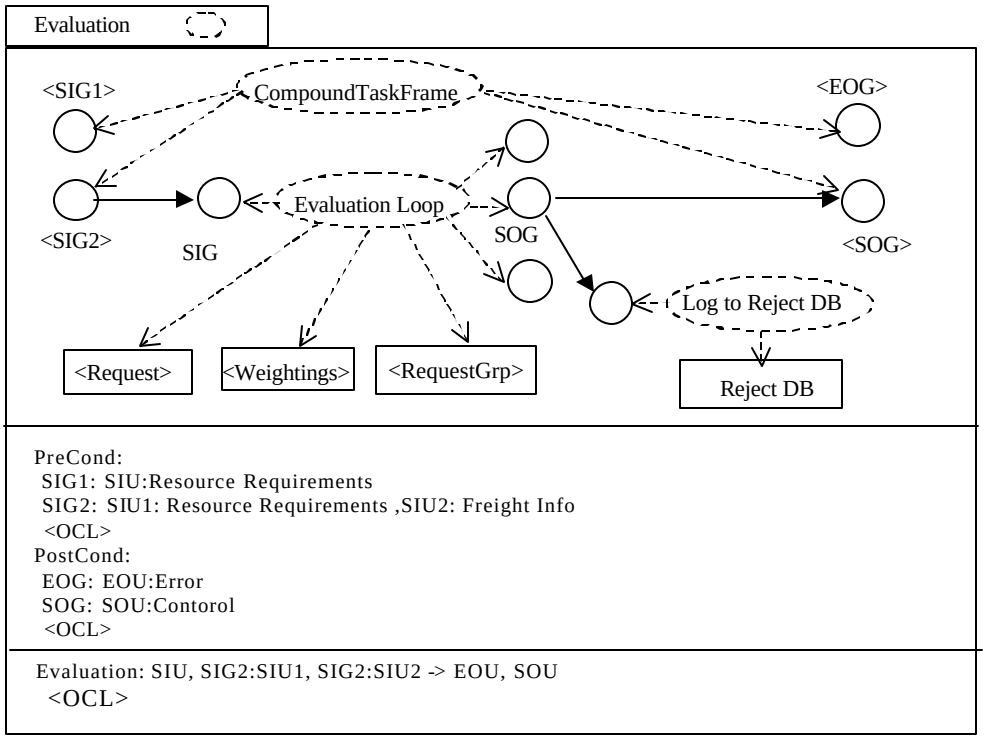
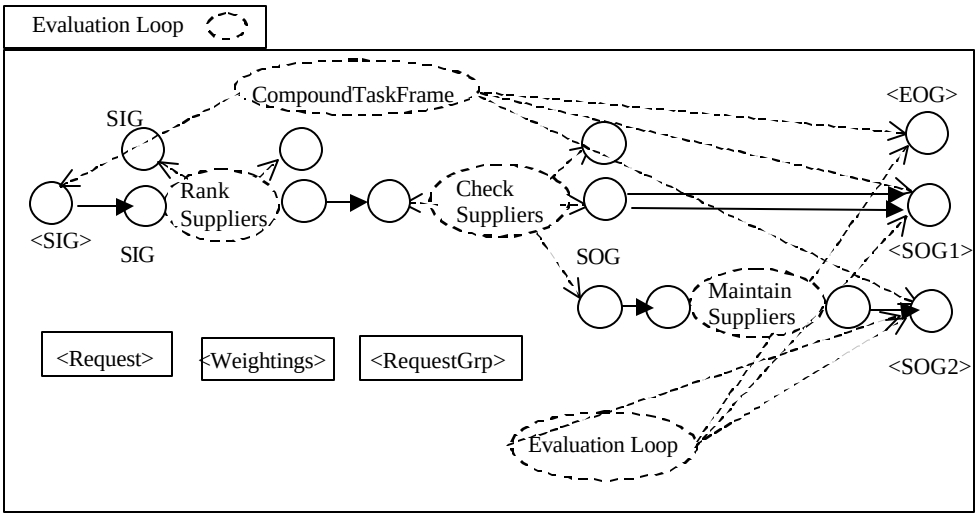


Figure 17 Procurement Pattern

2.11 Evaluation





Annex E - Technology mappings from EDOC to Distributed Component and Message Flow Platform Specific Models

Contents

List of Figures

List of Tables

1. Introduction to EDOC and Platform Specific Models

1.1 Introduction to EDOC

The ECA – Enterprise Collaboration Architecture is a model-driven architecture approach for specifying Enterprise Distributed Object Computing systems.

A forthcoming RFP will address technology mappings for "UML for EDOC".

This annex is non-normative and illustrates technology mappings to Distributed Component Models (in particular J2EE/EJB and CORBA/CCM) and a discussion on forthcoming mappings to Message Flow Models (FCM (Flow Composition Model from EDOC Part I), Workflow, MOM, WSDL, ...). This annex also contains a more detailed description of mappings from the EDOC Business Process profile to CORBA and FCM.

The EDOC vision is to provide a recursive collaboration based modeling approach that can be used at different levels of granularity, for both business and systems modeling. EDOC is able to support the specification of both loosely and tightly connected systems, with support for both synchronous and asynchronous communication in both container managed and message-based architectures.

This is done by providing a kernel in the CCA – the Component Collaboration Architecture with extensions for events, for entities, for relationships, and business processes and the use of patterns.

The focus of the ECA is on enterprise, computational and information specifications for a platform independent model of an EDOC system. These are transformed further to engineering and technology specifications for platform specific models using technology concepts from an appropriate Technology Specific Model.

Neither the business world, nor the computing world, applies only one paradigm to their problem space. Businesses use a combination of loosely coupled and tightly coupled processes, and computing solutions to deploy a combination of loosely coupled and tightly coupled styles of communication and interaction between distributed components.

An ECA based business process can be defined as event driven for some of its steps and workflow or request/response driven for others. Likewise, distributed components in the ECA can be configured to communicate with each other in a mixture of event-driven publish-and-subscribe, asynchronous peer-to-peer, and client-server remote invocation styles.

The EDOC Profile anticipates three levels of component coupling: linked, tightly coupled and loosely coupled.

Linked coupling refers to components that are co-located in the same address space. These components interact with each other directly, without communicating over a network. As such, they can interact without being identifiable over the network. Messaging will generally be synchronous, within the scope of a single transaction.

Tightly coupled components are distributed across multiple servers. These components will also interact with synchronous messaging, but messaging will occur over a network. While some messaging between the components may be asynchronous for performance and recoverability considerations, components are tightly coupled if any interactions between them are synchronous.

Loosely coupled components are distributed and only communicate asynchronously, through a messaging infrastructure. Communication is through messages and events. A message or event is issued in the scope of one transaction and accepted by one or more recipients in independent transactions. Messages and events are stored and forwarded. A message is a communicated with a defined recipient, and an event is a communicated (published) with self-declaring recipients (subscribers) unknown to the publisher.

The level of coupling between components has important performance and system flexibility implications. Generally, components should be designed in a level-of-coupling hierarchy so that components that are linked are within components that are tightly coupled, and tightly coupled components are then loosely coupled with each other. This coupling hierarchy should be reflected in the network accessibility property of components and the synchronous vs. asynchronous property of their ports.

With a consistent mapping to a particular technology, implementations of independently developed specifications should be operationally interoperable. Furthermore, components implemented with different technologies should be operationally interoperable if the technology mappings are consistent with the transformations provided by bridges between the technologies.

An EDOC computational specification can specify ProcessComponents at a number of different levels. These levels correspond to four general categories of ProcessComponent:

- E-Business Components
- Application Components
- Distributed Components
- Program Components

In this annex we only discuss a mapping of Process Components at the Distributed Component level, through mappings to EJB and CCM, even if the EDOC approach will be suitable for all levels. A wider scope for mapping is anticipated for a forthcoming RFP on technology mappings.

1.2 EDOC and Platform Specific Models

EDOC based specifications can be mapped down to various technology choices, and in particular both container-managed components and message-based services. Two Platform Specific Models are defined as part of the EDOC Profile, for Enterprise Java Beans and Java enterprise computing architectures, and for the Flow Composition Model (FCM).

The EJB metamodel captures the concepts that will be used to design an Enterprise JavaBean-based application down to the Java implementation classes. The metamodel includes the assembly and deployment descriptor

FCM is a general-purpose model that supports creating flow compositions of components and defining behaviors of those compositions using wiring diagrams. It provides a common set of technology abstractions across a variety of flow model types used in message brokering. FCM is related to the principal model in MQ-Series but it has more general applicability and is positioned as a layer of abstraction just above middleware technology,in contrast to the EDOC Business Processes profile which is intended technology neutral and intended for use in an analysis level model.

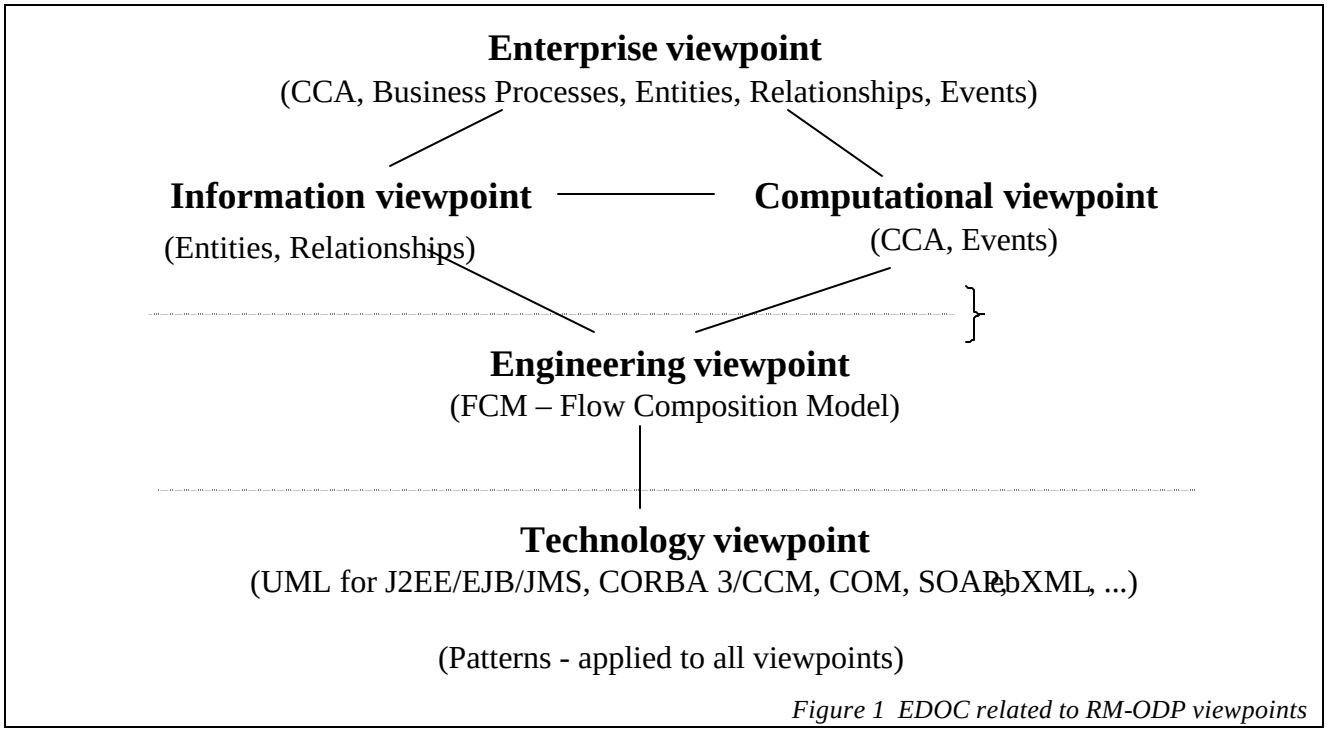


Figure 1 EDOC related to RM-ODP viewpoints

The focus of the main ECDOC parts is on implementation neutral enterprise, computation and information specifications. This is transformed further to platform specific technologies in the technology viewpoint, potentially with common platform abstractions in an engineering viewpoint, for instance with FCM – for message based platforms, and a similar abstraction for distributed component platforms.

Neither the business world, nor the computing world, applies only one paradigm to their problem space. Businesses use a combination of loosely coupled and tightly coupled processes and computing solutions deploy a combination of loosely coupled and tightly coupled styles of communication and interaction between distributed components.

The group of distributed component models includes technologies for container-manged components, such as EJB, CCM and COM+. The broader context for these technologies with J2EE, CORBA 3 and MS DNA/.Net also contains support for more asynchrounous and message-based services. EDOC based specifications can be mapped down to various technology choices, and in particular both container-managed components and message-based services.

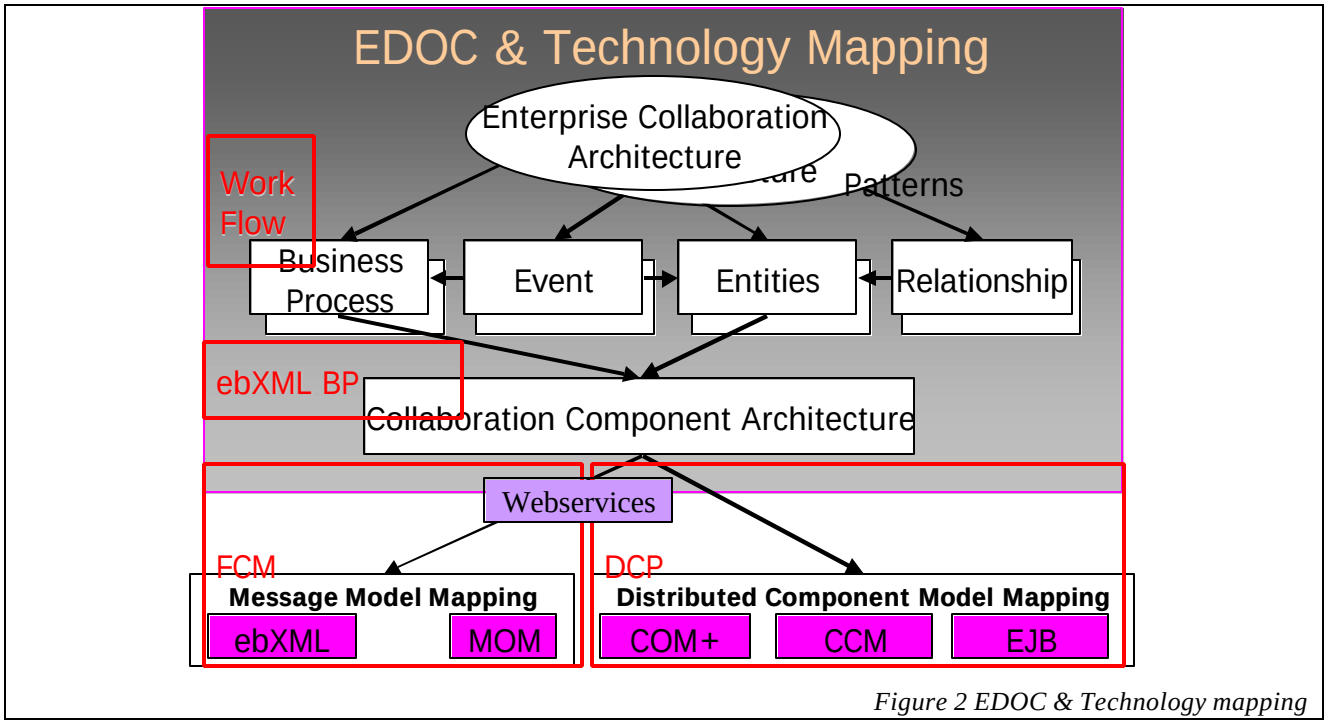


Figure 2 EDOC & Technology mapping

An EDOC based business process can be defined as event driven for some of its steps and workflow or request/response driven for others. Likewise, distributed components in the CCA profile can be configured to communicate with each other in a mixture of event-driven publish-and-subscribe, asynchronous peer-to-peer, and client-server remote invocation styles. The Event Model describes the purely event driven approach

The EDOC approach unifies specification for both Distributed Component and Message Oriented platform specific models. It includes a possibility for specifying both operations (request-response messages) and one-way messages and events. This might be mapped to Distributed Component technologies using traditional operations and one-way operations or event/notification services, and to Message Oriented technologies using composed messages (request-response pairs) and traditional messages. In broader

platform environments, such as CORBA 3 or J2EE, there is support for both technologies, and a combination of distributed components and messaging might be used in one system.

2. Principal Platform Specific Models

2.1 Distributed components using a multi-tier architecture model

To support flexible Enterprise systems, IT architectures can be structured as multi-tiered distributed architectures. As a reference model, a logical 4-tier architecture is presented .

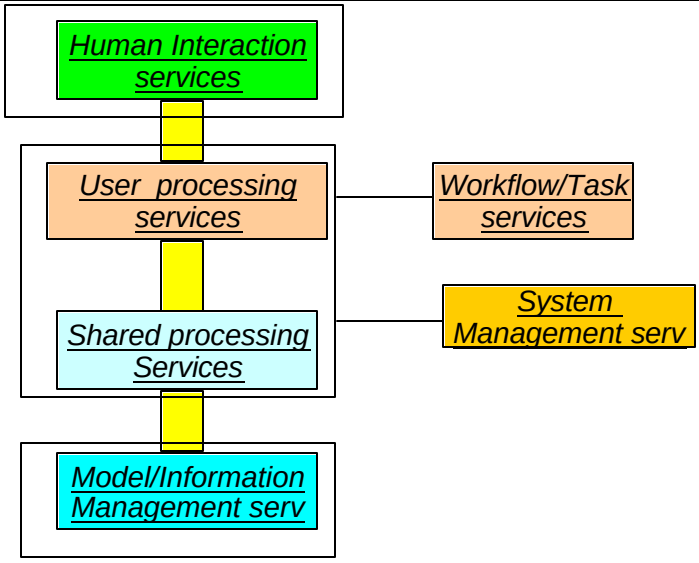


Figure 3 Multi-tiered reference model for Enterprise systems

The figure above shows the various parts of a multi-tiered reference model for enterprise systems:

- The *Human Interaction service* tier is responsible for physical interaction with the user, through display and input media, and an appropriate dialogue.
- The *Communication services* are responsible for connecting the various tiers together (although not labelled in the figure, the communication services are present as connections between the other service tiers).
- The *User processing service* tier is a part of the processing services responsible for the functionality required by the user
- The *Business processing service* tier is part of the processing services responsible for common services (both domain specific and general) that can be used by multiple users.
- The *Model/Information Management service* tier is responsible for physical data storage and data management.

- The *Workflow/Task services* is a set of services that can be viewed as a specialised processing service, supporting sequencing of actions and tools.

- The *System Management services* tier is orthogonal to the multi-tiered architecture, and might be introduced in multiple tiers.

The logical architecture can be mapped to multiple physical architectures. All tiers could be mapped into one monolithic application, or through different client-server architectures. The communication within and between the tiers can be inter- and/or intra- enterprise, and be both synchronous and asynchronous.

2.1.1 Implementation neutral technology model

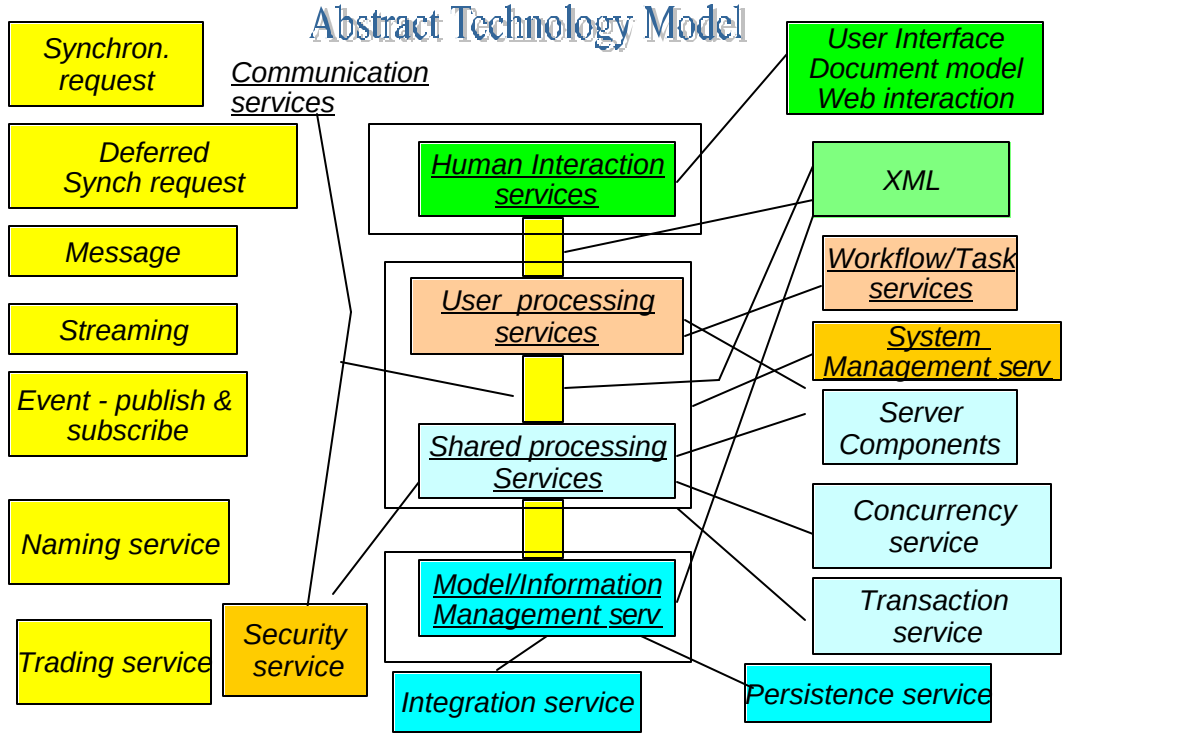


Figure 4 — Implementation neutral abstract technology model

The figure above shows an implementation neutral and abstract technology model of various services available in a typical target environment, such as CORBA 3 or J2EE.

The abstract technology model is related to typical service support for a multi-tiered service model, with the following services

Human Interaction Services are supported by various user interface mechanisms, including web-browsers.

Communication Services are supported by a principal set of 3+2 interaction modes: operations (synchronous or deferred synchronous requests), signals (event publish/subscribe or asynchronous messaging) and flows (with streaming). Associated with this is general naming and trading services.

In the context of UML for EDOC, it is important to note the need to support different communication mechanisms, in particular both synchronous and asynchronous communication, and both request/reply and notification oriented interaction.

Workflow/Task Services are supported by a set of workflow/task services.

System Management Services are supported by various kinds of user, application and security management services.

Processing Services are separated into user and shared services, where user services typically are supported in a single user mode, while shared services adds functionality for server-side and multi-user support with concurrency and transactions.

Model/Information Management Services are supported through data storage, persistence and manipulation services – potentially including various legacy system/format integration services.

The target infrastructure for the technology mapping of enterprise systems, will typicall consist of parts that supports the services described above.

In this document however, the focus for technology mapping is only on the container-managed shared processing services of J2EE/EJB and CORBA/CCM, thus addressing only a subset of a total system architecture. It is expected that a wider range of technologies will be targeted in a future RFP on technology mappings for EDOC.

2.2 Mapping to Distributed Component Models

EDOC and CCA supports both synchronous and asynchronous message specification. The current EJB, DCOM and CCM specifications are mostly tuned towards synchronous interactions. The current trend is, however, to integrate asynchronous message support als o for such managed components. The overall platform infrastructures already provides supports for asynchronous messaging through JMS, MSMQ and the CORBA Message service.

2.2.1 Mapping to CORBA 3 / CCM

This document contains a possible mapping from EDOC to the CCM – CORBA ComponentModel part of CORBA 3. A normative extension to this is expected to be submitted for a future RFP on EDOC technology mappings.

2.2.2 Mapping to J2EE/EJB

This document contains a possible mapping from EDOC to the EJB –Enterprise Java Beans part of J2EE. A normative extension to this is expected to be submitted for a future RFP on EDOC technology mappings.

2.2.3 Mapping to Microsoft COM+ and .Net

Future work could include a mapping to Microsoft COM+ and/or .Net.

2.3 Mapping to Message Flow Models

2.3.1 Mapping to the Flow Composition Model (FCM)

The FCM is a Flow Composition Model (FCM) that can describe the interactions and flows of information between application components

Further mapping from FCM to various message flow models will be addressed by the forthcoming "UML for EAI –Enterprise Application Integration" submission, and is thus not detailed further here.

2.3.2 Mapping to Workflow Services

Workflow is a possible service to support business processes. In the CORBA/CCM mapping it is shown how it is possible to map the EDOC Business Process profile to the CORBA Workflow service.

2.3.3 Mapping to Message Brokers

Message services are typically provided by Message Broker products, addressing the needs of business and application integration through management of information flow. It provides services that allow you to:

- Route a message to several destinations, using rules that act on the contents of one or more of the fields in the message or message header.
- Transform a message, so that applications using different formats can exchange messages in their own formats.
- Store and retrieve a message, or part of a message, in a database.
- Modify the contents of a message (for example, by adding data extracted from a database).
- Publish a message to make it available to other applications. Other applications can choose to receive publications that relate to specific topics, or that have specific content, or both.

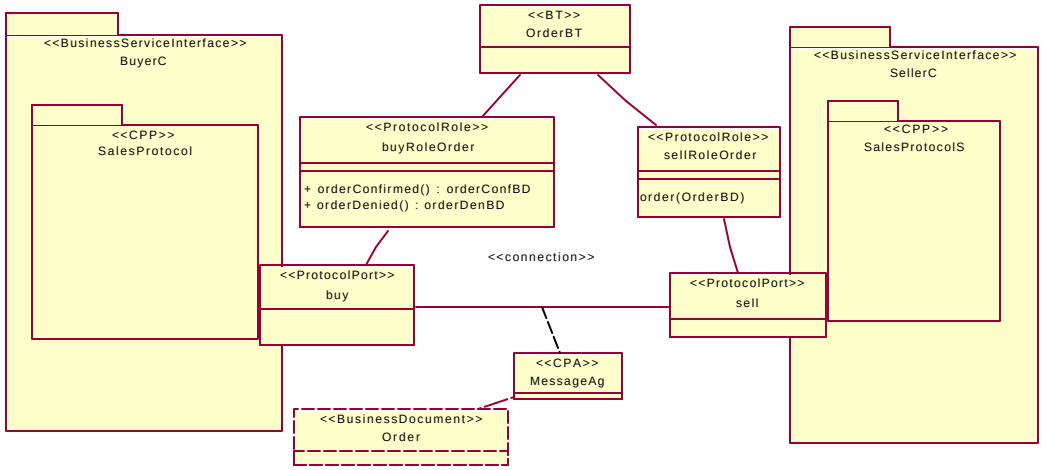
In particular in the EAI, Enterprise Application Integration, area it has been a trend towads the use of message brokers. A goal of EDOC has been to specify models that can provide a basis for EAI modeling, targeting infrastructure support from message brokers.

It is expected that the forthcoming RFP submission on UML for EAI will be able to make use of the base EDOC models.

2.3.4 Mapping to ebXML

The development of the concepts for EDOC and CCA has been developed in close contact with the development of the modeling approach for ebXML (ref. www.ebxml.org).

In particular the modeling approach has been tuned to meet the requirements for asynchronous messaging and business protocol specifications emerging in that environment.



CPP: Collaboration Protocol Profile
CPA: Collaboration Protocol Agreement
BT: Business Transaction
BD: Business Document

Figure 5 Mapping to ebXML

The ebXML message-based infrastructures is a potential target for EDOC-based specifications. The figure shows the mapping from the EDOC Buyer-Seller example to some of the related ebXML model concepts.

2.3.5 Mapping to Web Services

The emergence of Web Services, with specication languages such as WSDL (Web Services Definition Language) is a suitable target for EDOC based models. It is expected that Web Services will be a target for a future RFP for EDOC technology mappings.

2.4 Mapping approaches

A mapping to a technical platform can use one of two basic approaches:

Type 1. It can describe how to transform a model to a set of declarations expressed in the native declarative language of the chosen technical platform. This kind of transformation targeted to the CORBA platform generates declarations expressed in CORBA IDL, i.e. CORBA interfaces, valuetypes, etc. If targeted to the Java platform, it generates declarative Java code, i.e. Java interfaces and abstract classes. If targeted to XML, it generates an XML DTD or XML Schema, both of which are essentially declarative code.

Type 2. It can describe how transform a model to another UML model expressed in terms of a UML profile targeted to the chosen technical platform, such as the UML Profile for CORBA[1]

[1] [UML-CORBA]

or the UML Profile for EJB[2]. Such UML profiles support expression via UML of declarative semantics in terms of the concepts native to the chosen technical platform.

This document discusses the mapping from EDOC to J2EE/EJB and CORBA/CCM using a *Type 2* approach with a discussion on mapping from UML for EDOC model elements to elements of the JSR-26 UML for EJB profile, and to an early draft UML for CCM profile.

3. Mapping from EDOC to J2EE/EJB

3.1 The Model of J2EE

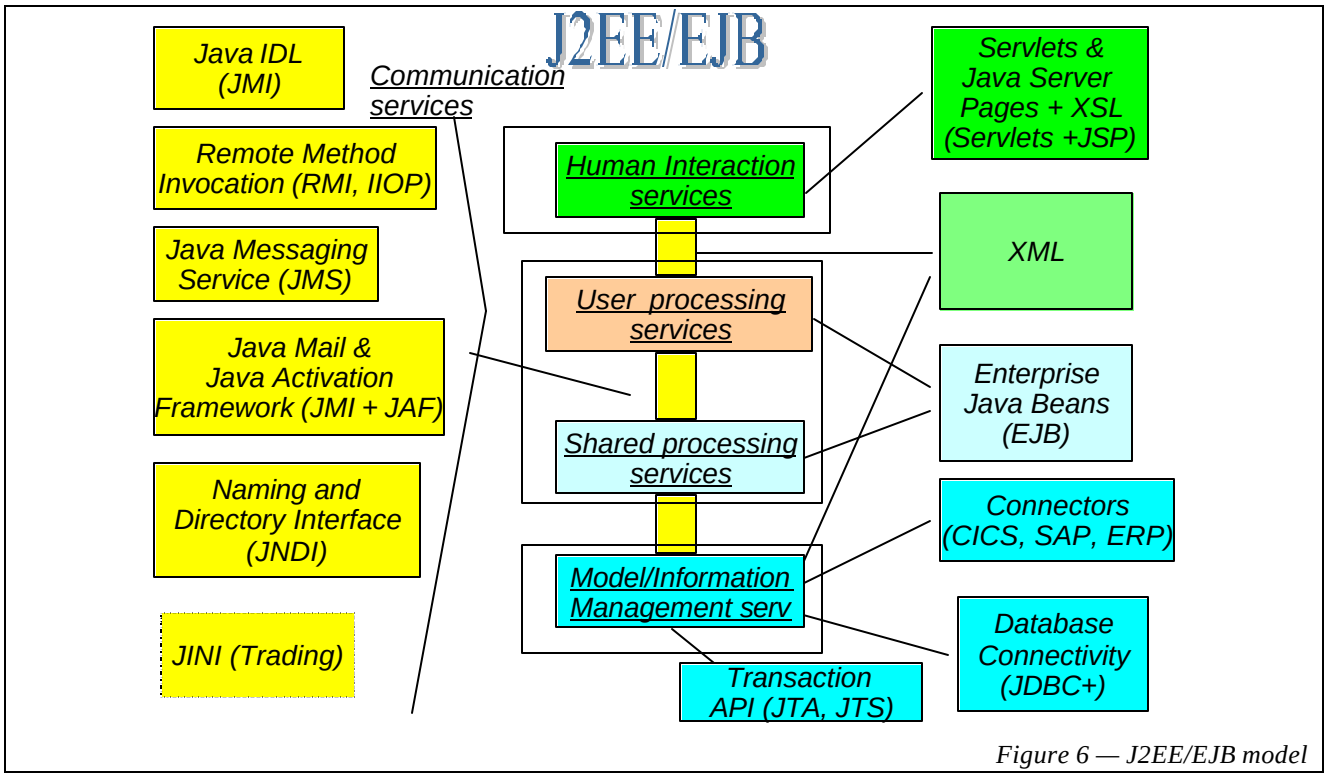


Figure 6 — J2EE/EJB model

Human Interaction Services are supported by the Java windowing system and through support for web-browsers, typically with web-server support such as Java Server Pages (JSP) and increased support for XML and XSLT.

Communication Services are supported by Java RMI, as well as with the Java messaging service and event&notification through the messaging service. Further support for communication of XML-structures will be provided by the Java XML-API.

Workflow/Task Services are not supported directly.

System Management Services are supported by Java Security and associated user services.

[2] [JSR-26]

Processing Services are supported by server-side Enterprise Java Beans (EJB) and the associated concurrency and transaction service.

Model/Information Management Services are supported by the JDBC and a forthcoming Java persistence service, as well as the current Java serialisation.

In the forhcoming mapping presented here we are only focusing on the mapping of shared processing services, using the EJB technology.

3.2 Model and UML Profile for EJB

3.2.1 A basis in UML for EJB: JSR-26

The EJB and Java Metamodel used here is based on the public version of the "UML for EJB" profile, JSR-26, emerging from the Java Community Process. Some input to discussions on this is described in the part I of this submission, UML for EDOC Part I – Chapter 5 section 1.

3.2.2 EJB Design Model – External view

3.2.2.1 UML Stereotypes

Stereotype	Applies to	Definition
<<EJBCreateMethod>>	Operation	Specializes «EJBHomeMethod». Indicates that the Operation represents an EJB Create Method.
<<EJBFinderMethod>>	Operation	Specializes «EJBHomeMethod». Indicates that the Operation represents an EJB Finder Method.
<<EJBRemoteMethod>>	Operation	Indicates that the Operation represents an EJB Remote Method.
<<EJBRemoteInterface>>	Class	Specializes the standard UML Stereotype «type». Indicates that the UML Class represents an EJB Remote Interface.
<<EJBHomeInterface>>	Class	An abstract Stereotype indicating that the UML Class represents an EJB Home Interface. Specializes the standard UML Stereotype «type».
<<EJBSessionHomeInterface>>	Class	Indicates that the Class represents an EJB Session Home. Specializes the Stereotype «HomeInterface».
<<EJBEntityHomeInterface>>	Class	Indicates that the Class represents an EJB Entity Home. Specializes the Stereotype «HomeInterface».
<<EJBPrimaryKey>>	Usage	Indicates that the supplier of the Usage represents the EJB Primary Key Class for the EJB Entity Home represented by the client.

Table 1: EJB Design Model – External View - UML Stereotypes

3.2.2.2 UML Tagged Values

Tagged Value	Applies To	Definition
EJBSessionType	Class <<EJBSessionHomeInterface>>	Stateful or Stateless. Indicates whether or not the EJB Session Bean maintains state.

Table 2: EJB Design Model - External View - UML Tagged Values

3.2.3 EJB Design Model – Internal view

3.2.3.1 UML Stereotypes

Stereotype	Applies to	Definition
<<EJBCmpField>>	Attribute	Indicates that the Attribute represents a container-managed field for an EJB Entity Bean with container-managed persistence
<<EJBPrimaryKeyField>>	Attribute	Specializes «EJBCmpField». Indicates that the Attribute is the primary key field for an EJB Entity Bean with container-managed persistence.
<<EJBRealizeHome>>	Abstraction	Indicates that the supplier of the Abstraction represents an EJB Home Interface for the EJB Implementation Class represented by the client.
<<EJBRelizeRemote>>	Abstraction	Indicates that the supplier of the Abstraction represents an EJB Remote Interface for the EJB Implementation Class represented by the client.
<<EJBImplementation>>	Class	Specializes the standard UML Stereotype «implementationClass». Indicates that the Class describes an EJB Implementation Class, distinguishing it from other Classes that may appear within a UML Subsystem that represents an EJB Enterprise Bean.
<<EJBEnterpriseBean>>	Subsystem	An abstract Stereotype indicating that the Subsystem represents an EJB Enterprise Bean.
<<EJBSessionBean>>	Subsystem	Indicates that the Subsystem represents an EJB Session Bean. Specializes «EJBEnterpriseBean».
<<EJBEntityBean>>	Subsystem	Indicates that the Subsystem represents an EJB Entity Bean. Specializes «EJBEnterpriseBean».
<<EJBReference>>	Association	A Stereotype indicating that the navigable end of the UML Association represents a referenced EJB Enterprise Bean.
<<EJBAccess>>	Association	A Stereotype indicating that the UML Association defines a security role name relationship between a UML Actor and an «EJBEnterpriseBean».

Table 3: EJB Design Model – Internal View - UML Stereotypes

3.2.3.2 UML Tagged Values

Tagged value	Applies to	Definition
EJBRoleNames	Operation	A comma-delimited list of Strings, designating the security roles that may invoke the Operation.
EJBTransAttribute	Operation	An enumeration with values Not Supported, Supports, Required, RequiresNew, Mandatory, or Never. Defines the transaction management policy for the Operation.
EJBEnvEntries	Subsystem <<EJBEnterpriseBean>>	A comma-delimited list of tuples, designating the environment entries used by the EJB Enterprise Bean, of the form <name, type, value>.
EJBNameInJAR	Subssytem <<EJBEnterpriseBean>>	The name used for the EJB Enterprise Bean in the EJB-JAR. Defaults to the name of the EJB Remote Interface.
EJBReferences	Subssytem <<EJBEnterpriseBean>>	A comma-delimited list of tuples, designating the other EJB Enterprise Beans referenced by the EJB Enterprise Bean, of the form <name, type, home, remote>.
EJBResources	Subssytem <<EJBEnterpriseBean>>	A comma-delimited list of tuples, designating the resource factories used by the EJB Enterprise Bean, of the form <name, type,auth>.
EJBSecurityRoles	Subssytem <<EJBEnterpriseBean>>	A comma-delimited list of tuples, designating the role names that may invoke ALL operations on the EJB Enterprise Bean, of the form <name, link>.

Tagged value	Applies to	Definition
EJBTransType	Subssytem <<EJBSessionBean>>	An enumeration with values Bean or Container. Indicates whether the transactions of the EJB Session Bean are managed by the EJB Session Bean or by its container, respectively.
EJBPersistenceType	Subssytem <<EJBEntityBean>>	An enumeration with values Bean or Container. Indicates whether the persistence of the EJB Entity Bean is managed by the EJB Entity Bean or by its container, respectively.
EJBReentrant	Subssytem <<EJBEntityBean>>	A Boolean value indicating whether or not the EJB Entity Bean can be called reentrantly.

Table 4: EJB Design Model – Internal View – Tagged values

3.2.4 EJB Implementation Model

3.2.4.1 UML Stereotypes

Stereotype	Applies to	Definition
<<EJB-JAR>>	Package	Specializes the Stereotype «JavaArchiveFile». Indicates that the Package represents an EJB-JAR.
<<EJBDescriptor>>	Component	Specializes the standard Stereotype «file». Indicates that the Component represents an EJB Deployment Descriptor.
<<EJBClientJAR>>	Usage	Indicates that the client of the Usage represents an ejb-client-jar for the EJB-JAR represented by the supplier of the Usage.

Table 5: EJB Implementation Model – UML Stereotypes

3.3 Mapping from the EDOC CCA Profile

This section details the mappings from the EDOC Part I CCA, Component Collaboration Architecture.

3.3.1 Mapping Process Components and Protocols

Part of a component's specification is the set of **protocols** it implements, a protocol specifies what messages the component sends and receives when it collaborates with another component and the **choreography** of those messages – when they can be sent and received. Each protocol the component supports is provided via a "**port**", the connection point between components.

Protocols, ports and choreography comprise the contract on the outside of the component. Protocols are also used for large-grain interactions, such as for B2B components.

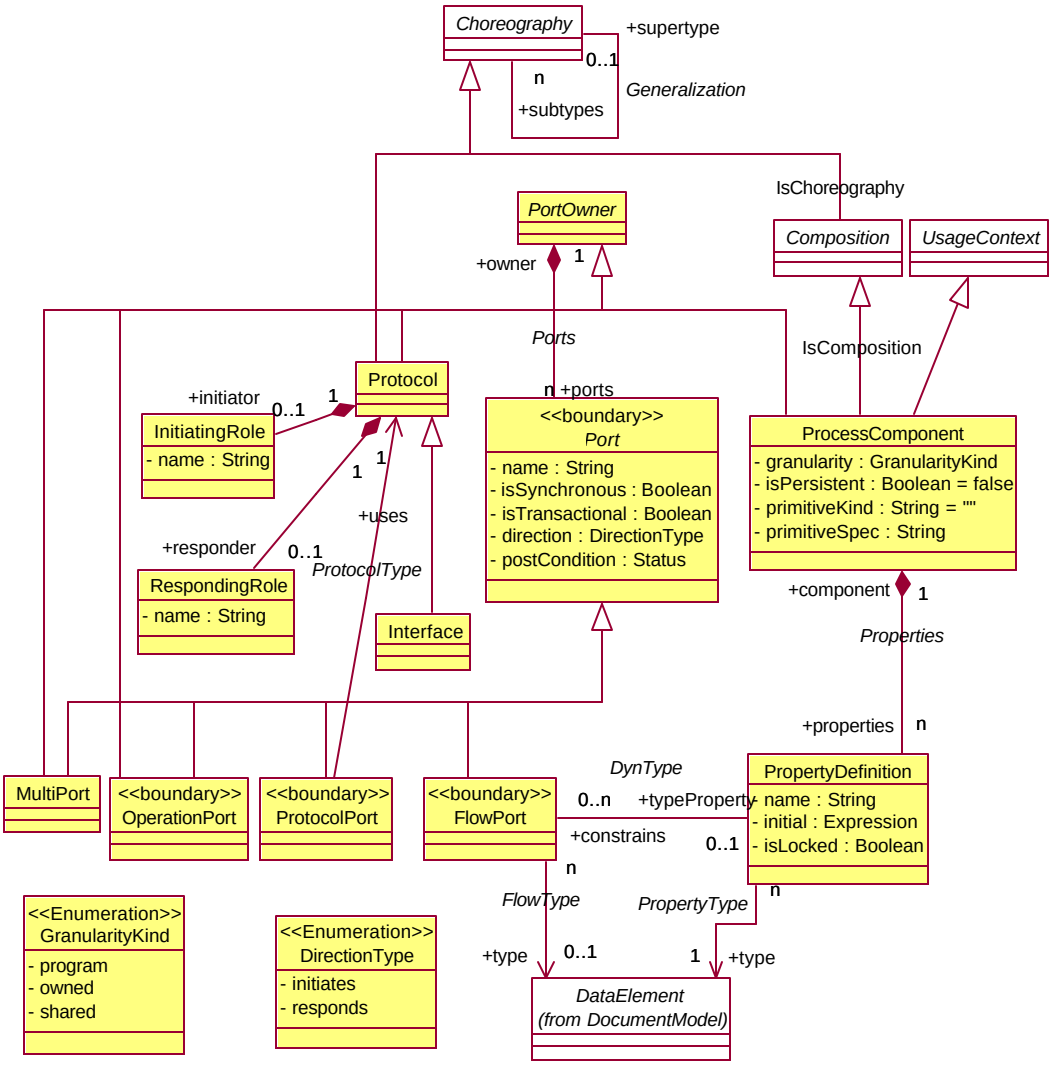


Figure 7 CCA Structural Specification Model

In mapping to Distributed Component technologies, such as CCM, EJB, COM+ the most significant specification aspect are the interfaces. In CCA interfaces are specified implicitly through the protocol messages being sent from a Process Component and the messages being received as replies. If the interaction is taking place through a synchronous interaction, the operations of the interface can be derived through combining the request and corresponding reply messages into the signature of an operation of the required interface.

In mapping ProcessComponents and associated Ports with interactions according to Protocols to an Interface-oriented technology, there is an issue of how to map FlowPorts and OperationPorts, and how to combine these into interfaces. The EDOC Protocol structure specification specifies messages being sent both ways between the protocol initiator and the responder. In an interface-oriented mapping of this it is necessary to analyse the direction of messages with respect to being sent from initiator or responder. The

rolename of the protocol can be used to give a default name for the interface to be provided at each side, if the protocol represents a dual interface situation. The interface will be populated by the operationport and flowports (operations without result) going in the respective directions.

The EDOC model allows for ProcessComponents with multiple ports, and thereby both using and providing multiple interfaces. Since some models, i.e. EJB allows for only one interface per component (Bean) the interfaces related to multiple ports need to be merged, with a potential for name-clashes and name-conflicts.

In the Composition model it is possible to delegate (connect) message flow from one component to another, but still provide the message reply back to the initiator. This requires that a reference to the reply-to object is included in the argument of a message, or that the infrastructure provides for some means to identify the caller.

Subprotocols can be initiated by providing a reference to an object with an interface according to its role in the subprotocol.

Metamodel element name	**Map comment**	**EJB**
ProcessComponent	A ProcessComponent represents the contract for a component that performs actions – it "does something". A ProcessComponent may realize a set of Ports for interaction with other ProcessComponents and it may be configured with properties	Maps to an <<**EnterpriseBean**>> (in this description) but can also naturally map to higher (Business Process) or lower (Object) level concepts. (One of EJBSessionBean, or EJBEntityBean)
IsPersistent (Property of ProcessComponent)	default=false, if true stores session specific state across interactions	If true, EJBSessionType = Stateful. If false EJBSessionType = Stateless (default) Is always true for EJBEntity.
Port	A port realizes a simple or complex conversation for a ProcessComponent or protocol. Each port is connected with collaborative components that speak the same protocol All interactions with a ProcessComponent are done via one of its ports Each port provides a connection point for interaction with other components or services and realizes a specific protocol. The protocol may be simple and use a "FlowPort" or the protocol may be complex and use a "ProtocolPort" or an "OperationPort". If allowed by its protocol, a port may send and receive information.	Mapping depends on kind of port. See below. External view: <<**EJBRemoteInterface**>> Internal view: <<**EJBImplementation**>> with <<**EJBRealizeRemote**>> and <<**EJBRealizeHome**>>
IsTransactional (Property of port)	interactions with the component are transactional & atomic	IsTransacational = true means a mapping to an EJBTransAttribute of one of supports, required, requiresNew, mandatory IsTransacational = false means a mapping to EJBTransAttribute of one

Metamodel element name	**Map comment**	**EJB**
		of notSupporte or Never
IsSynchronous (Property of port)	A port may interact synchronously or asynchronously. A port that is marked as synchronous is required to interact using synchronous messages and return values	IsSynchronous = true means mapping to operations, IsSynchronous = false (default) means mapping to a message (JMS) or an event notification mechanism
Direction (Property of port) Initiates or Responds	Indicates that the port will either initiate or respond to the related type. An initiating port will send the first message. Note that by using ProtocolPorts a port may be the initiator of some protocols and the responder to others.	The direction of the port is with respect to the protocol, it is only implicitly mapped to EJB (See below)
FlowPort	A Flow Port is a port which defines a data flow in or out of the port on behalf of the owning component or protocol. direction in is one way operation or out is a one way call	Direction *Initiates*: Maps to a **message being sent**. Direction *Responds*: Maps to a **message being received**. In the synchrounous case this is a mapping to an operation without return values in an interface, In the asynchrounous case this is a mapping to an event being received.
ProtocolPort	A protocol port is used for potentially complex two-way interactions between components	Maps to the **Interface** and/or **Message/Event** being used and/or provided for a two-way interaction
MultiPort	Each port owned by the MultiPort will "buffer" information sent to that port until all the ports within the MultiPort have received data, at this time all the ports will send their data	Direction *Responds*: Maps to an implementation where a set of messages (events) need to be received by the port before it is sent further.
OperationPort	An operation port represents the typical call/return pattern of an operation. The OperationPort is a PortOwner which is constrained to contain only flow ports, exactly one of which must have its direction set to "initiates".	Direction *Initiates*: This maps to the **invoker of an operation**, which is not explicitly represented in EJB. Direction *Responds*: This maps to an **operation to be implemented**. (an operation in an interface) (EJBRemoteInterface).

Metamodel element name	Map comment	EJB
Protocol	A protocol defines a type of conversation between two parties, the initiator and responder. One protocol role is the initiator of the conversation and the other the responder. However, after the conversation has been initiated, individual messages and sub-protocols may by initiated by either party.	Maps to a **description of the messages and operation interactions in a conversation**. Only the responding side is explicitly represented in EJB.
Interface	An interface is a protocol constrained to match the capabilities of the typical object interface. It is constrained to only contain OperationPorts and FlowPorts and all of its ports must respond to the interaction (making interfaces one-way) Each OperationPort or FlowPort in the Interface will map to a method. A ProtocolPort which initiates the Interface will call the interface. A ProtocolPort which Responds will implement the interface Existing interface	An EDOC interface represents a protocol that maps directly to a UML interface that again is mapped to a **Java (EJB) interface**. Depending on the direction (initiates or responds) the Interface is either used or provided (responds).
InitiatingRole	The role of the protocol which will send the first message Default Interface name	Represents the **EJB Bean** that is the initiator of a protocol
RespondingRole	The role in the protocol which will receive the first message Default Interface name	Represents the **EJB Bean** that is the receiver of the first message in a protocol
PropertyDefinition	PropertyDefinition defines name and type for properties which may be set when the ProcessComponent is used	Properties on the **EJB Bean**. Can be mapped to EJBEnvEntries

Table 6: Stereotypes for Structural Specification (UML notation: Class Diagram)

A protocol specifies the conversation between two ProcessComponents (via their ports). Each component that is using that protocol must use it from the perspective of the "initiating role" or the "responding role". Each of these components will use every port in the protocol, but in complementary directions.

Each port is connected with collaborative components that speak the same protocol. Multi-party conversions are defined by components using multiple ports, one for each kind of party.

Components interact with their environment through ports. A port has a defined interaction protocol. Ports may send messages, receive messages, or both. A port may be implemented as an object interface, e.g., CORBA or Java interface.

Ports are synchronous or asynchronous. A synchronous port communicates within the context of a transaction. An asynchronous port communicates in a store-and-forward manner so that sending a message occurs in the context of one transaction and receipt of the message then occurs in the context of another transaction.

Ports may communicate with messages or event notices. A message is directed to a specific destination. An event notice is published to the communication infrastructure to be delivered to subscribers—destinations that have expressed interest. The messages and event notices may be communicated synchronously or asynchronously.

All Data Managers will have a synchronous interface port that represents the typical object interface. A Data Manager may have other ports, such as to send messages to other Data Managers, and to send and receive asynchronous messages and events.

3.3.2 Mapping Composition

Components may be abstract (having only an outside) or concrete (having an inside and outside). Frequently a concrete component inherits its external contract from an abstract component – implementing that component.

There may be any number of implementations for an **abstract component** and various ways to "bind" the correct implementation when a component is used.

The two basic kinds of concrete components are:

- **Primitive components** – those that are built with programming languages or by wrapping legacy systems.

- **Composed components** – Components that are built from other components; these use other components to implement the new components functionality. Composed components are defined using a **composition**.

Compositions define how components are *used*. Inside of a composition components are used, configured and connected. This connected set of component usage's implements the behavior of the composition in terms of these other components – which may be primitive, composed or abstract components.

Compositions can also include a **choreography** of how the components used work together, which should execute when.

Compositions are used to build composed components out of other components and to describe community processes – how a set of large grain components works together for some purpose.

Central to compositions are the **connections** between components, values for **configuration properties** and the ability to **bind** concrete components to a component usage.

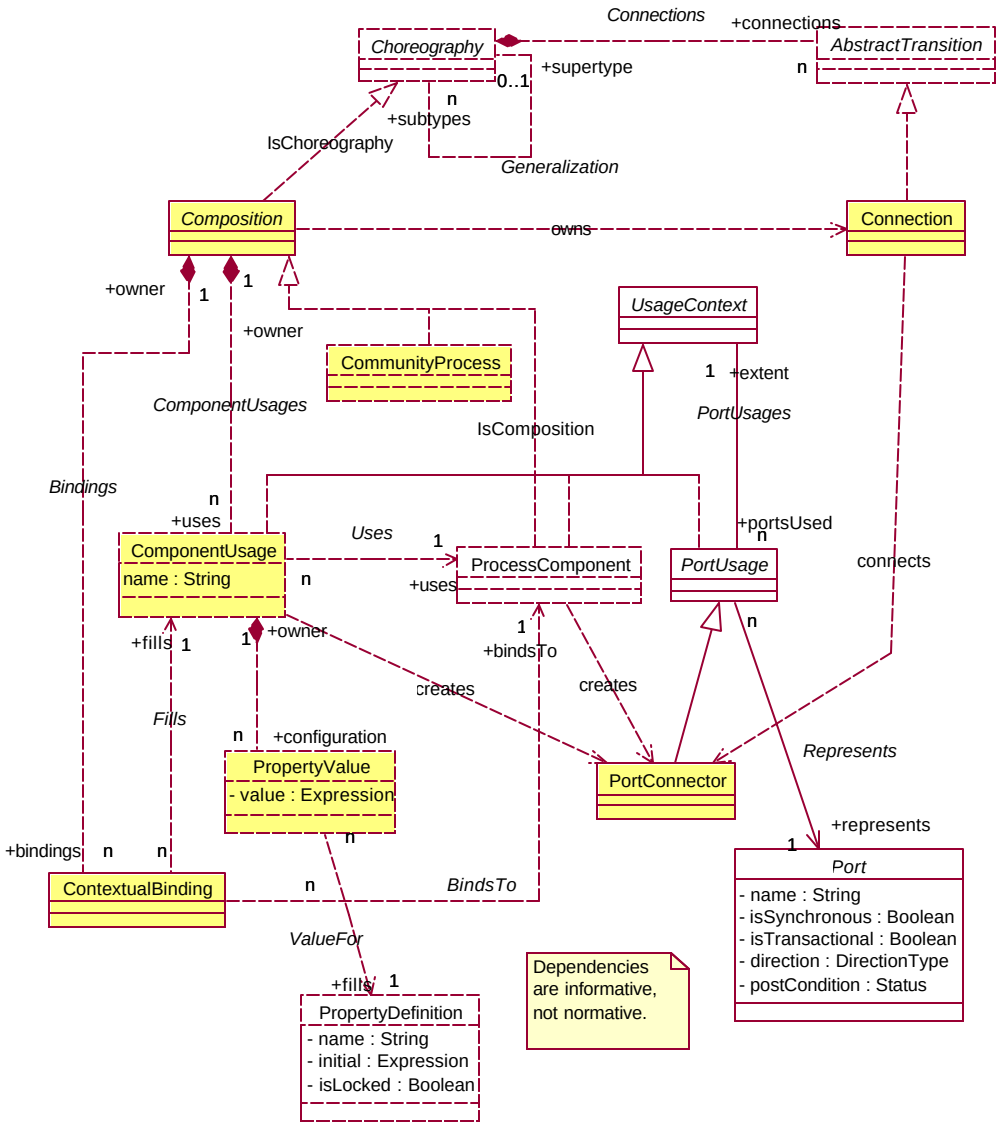


Figure 8 CCA Composition and usage model

Metamodel element name	Mapping description	EJB
Composition	Compositions describe how instances of ProcessComponents (called ComponentUsages) are configured (with PropertyValues and ContextualBindings) and connected (with Connections) to implement the composed ProcessComponent or CommunityProcess.	Maps to the **structure of interacting components** (beans), and how they are configured and connected. Not explicitly represented in EJB.
ComponentUsage	A ComponentUsage will cause a ProcessComponent instance to be created at runtime (this instantiation may be real or virtual).	Maps to the **use of one component by another**.
PortConnector	PortConnector provides a "connection point" for ComponentUsages within a composition and exposes the defined ports within the composition. The connections between PortConnectors are made with Connections.	Maps to the realisation of ports. **EJB interfaces** represents responding ports. Initiating ports are not represented in EJB.
Connection	A Connection connects two PortConnectors within a composition. Each port can produce and/or consume message events. The connection logically registers each port connector as a listener to the other, effectively making them collaborators.	This is the representation of the communication between two PortConnectors, either the communication link for operation invocations, or a mechanism for message event handling. (Event handling is not directly supported in EJB 1.1)
PropertyValue	a ProcessComponent may have configuration properties –which are defined by a PropertyDefinition. When the component is used in a ComponentUsage those properties values may be set using a PropertyValue.	The values of **properties defined by name/type**, e.g as EJBEnvEntries
ContextualBinding	Contextual Binding allows the substitution of a more concrete ProcessComponent for a compatible abstract ProcessComponent when an abstract composed ProcessComponent is used.	Interface conformance allows for **multiple implementations** of a bean.
CommunityProcess	CommunityProcess may be thought of as the "top level composition" in a CCA specification, it is a specification of a composition of ProcessComponents that work together for some purpose other than specifying another ProcessComponent.	**Highest level of component interaction**

Table 7: Stereotypes for Composition (UML notation: Collaboration Diagram at specification level)

Specifications of composition can be used to automatically create components that use existing components, and support this dynamically or through code generation.

3.3.3 Mapping Choreography

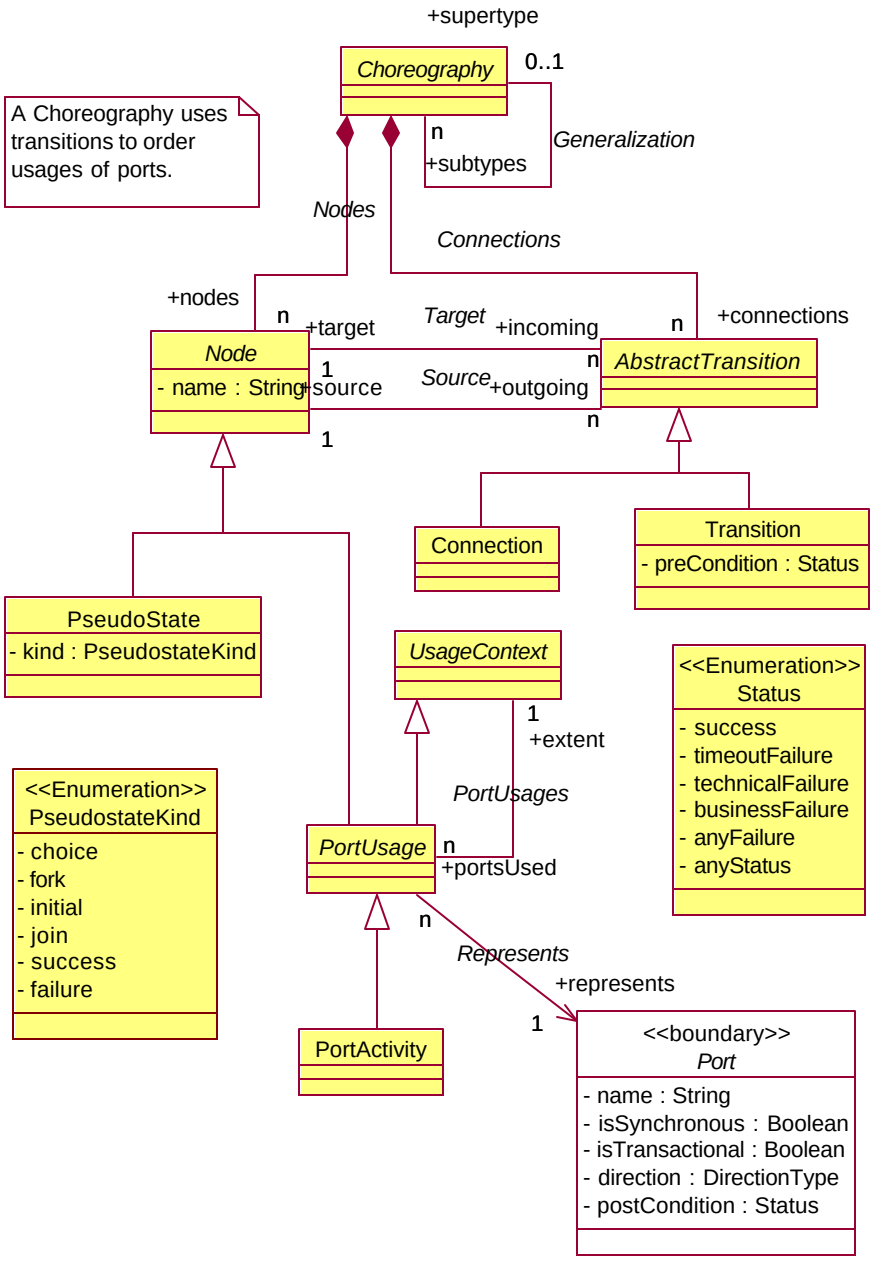


Figure 9 CCA Choreogrpahy

A Choreography specifies how messages will flow between PortUsages. The choreography may be externally oriented, specifying the contract a component will have with other

components or, it may be internally oriented, specifying the flow of messages within a composition. External choreographies are shown as an activity graph while internal choreography is shown as part of a collaboration. An external choreography may be defined for a protocol or a ProcessComponent.

A **Choreography** uses **Connections** and **transitions** to order port messages as a state machine. Each "node" in the choreography must refer to a state or a port usage.

Choreography is an abstract capability that is inherited by ProcessComponents and protocols.

Choreography may be used at multiple levels;

- A protocol Choreography specifies the sequencing of messages and sub-protocols between protocol roles. This is much like a sequence diagram.
- A process component Choreography specifies the sequencing of multiple messages and sub-protocols of ports and is part of the external contract of the component.

The use of choreography at all of these levels is not always required, as sufficient specification may be determined from the other layers.

Specifications of choreography can be used to automatically create executable code, or to support the execution of a workflow engine, in the case of a mapping to a coordinating workflow system.

3.3.4 Mapping Document Model

The information that flows between components is described in a **Document Model**, the structure of information exchanged. The document model also forms the basis for information entities and a generic **information model**. The information model is acted on by CCA process components.

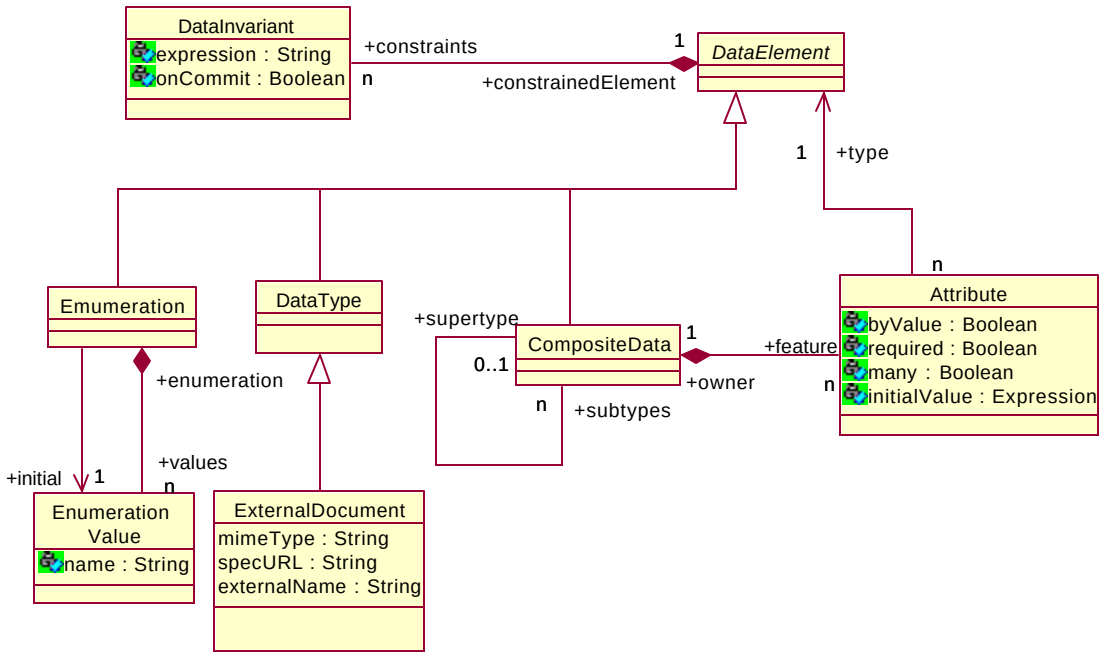


Figure 10 CCA Document Model

A **data element** represents a type of data which may either be primitive **DataTypes** or composite**. CompositeData** has named attributes which reference other types. Any type may have a **DataInvariant** expression.

Attributes may be **isByValue**, which are strongly contained or may simply reference other data elements provided by some external service. Attributes may also be marked as **required** and/or **many** to indicate cardinality. **DataTypes** define local data – these types are defined outside of CCA. **ExternalDocument** defines a document defined in an external type system. An **enumeration** defines a type with a fixed set of values

Metamodel element name	**Mapping Comment**	**EJB**
CompositeData	A datatype composed of other types in the form of attributes	Maps to a **data representation** as being used for a set of actual arguments in a message/operation or in a generic data structure
ExternalDocument	A large, self contained document defined in an external type system such as XML, Cobol or Java that may or may not map to the ECA document model	Mapping to an external datastructure to be interpreted.

Metamodel element name	Mapping Comment	EJB
DataInvariant	A constraint on the legal values of a data element.	Constraint that could be checked by generated code.
DataType	Data types may have their structure and semantics defined outside of CCA. The following data types are defined for all specializations of CCA: String, Integer, Float, Decimal, Boolean.	Maps to the **basic datatypes of Java and EJB**. The definition of the EDOC platform independent basic datatypes and the rules for mapping to Java/EJB will be done according to the rules being defined by MOF 1.4 currently in progress.
Enumeration	An enumeration defines a type that may have a fixed set of values.	Maps according to MOF 1.4 as above.
Attribute	Defines one "slot" of a composite type that may be filled by a data element of "type".	Maps to the definition of **one attribute in CompositeData**.

Table 8: Stereotypes for DocumentModel (UML notation: Class Diagram)

Specifications of data elements from the document model are used for creating value objects for messages, and for defining data representations.

3.4 Mapping from the Entities Profile

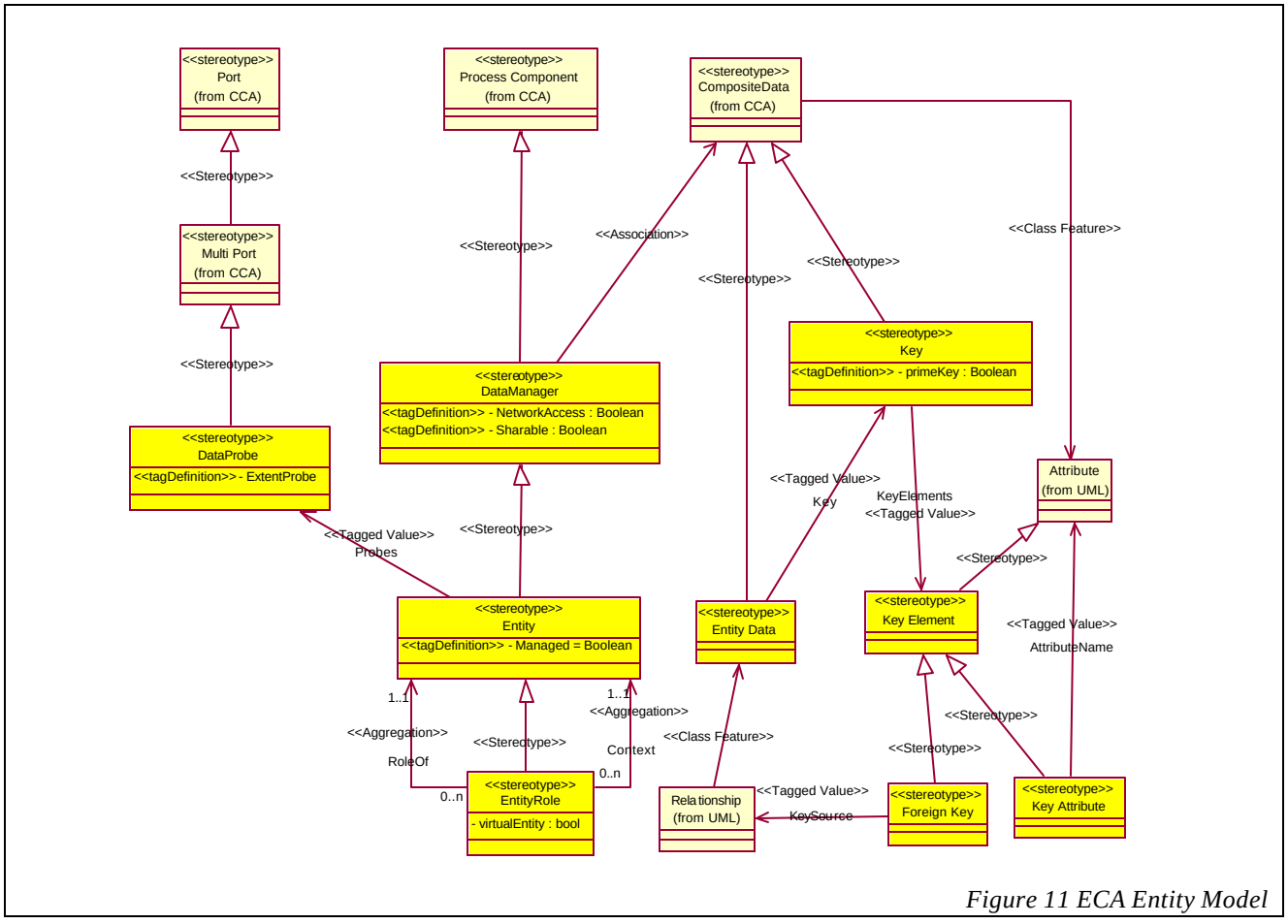


Figure 11 ECA Entity Model

Data Managers are components which manages a given type of data. **Entities** are identifiable elements in an information model which are managed by **Entity Managers**. **Keys** provide the identity for entities.

Metamodel Element	**Mapping comment**	**EJB**
Data Manager	Data Manager is a functional component that provides access to and may perform operations on its associated Composite Data (i.e., its state). The Data Manager defines ports for access to operations on the state data	Maps typically to an **EJBEntityBean**, but might sometimes be handled by a stateful EJBSessionBean. If not network-addressable (see below) this might be handled by a dependent object.
Network-addressable	A Boolean value which indicates if the Data Manager is intended to be accessible over the network.	If true, maps the DataManager to an **EJBSessionBean or EJBEntityBean**.
Shareable	Boolean value which indicates if the Data Manager can be shared by multiple transactions/sessions. A Data Manager that is not sharable is either transient or depends on a sharable Data Manager that contains it for persistence.	If true, maps the DataManager to an **EJBEntityBean**.
Entity	Entity is an object representing something in the real world of the application domain. It incorporates Entity Data that represents the state of the real world thing, and it provides the functionality to encapsulate the Entity Data and provide associated business logic.	Maps to an **EJBEntityBean** (if *Shareable* and/or *Network-addressable, or Managed)* if not to **a dependent object**.
Managed (Entity Property)	Boolean value that indicates if the Entity type is *managed*. If it is managed, then the implementation provides a mechanism for accessing the extent of all instances	If true implies the declaration of an **EJBHomeInterface**.
Entity Data	Entity Data is the data structure that represents a concept in the business domain. It is equivalent to an entity in data modelling or a relation in a relational database. In a Data Manager or its specializations, such as Entity, it represents the state of an object.	Maps to the data representation part of an **EJB EntityBean.**
Entity Role	Entity Role extends its parent Entity for participation in a particular context. An Entity may have a number of associated Entity Roles reflecting participation in multiple contexts	Maps to another associated **EJBEntityBean** or dependent object
Virtual Entity	Boolean value that indicates if the Entity Role incorporates and extends the primary interface of the parent Entity it represents, i.e., it can be used in place of the primary Entity	Maps to another associated **EJBEntityBean or dependent object**, extending the primary interface.

Metamodel Element	Mapping comment	EJB
Key	Key is composed of key elements which may be selected attribute values of the associated Entity Data or Foreign Keys	Maps to an **EJBPrimaryKey**.
Foreign Key	A Foreign Key is the key of a related Entity Data.	Maps to **EJBPrimaryKey** for another EJBEntityBean
Data Probe	Data Probe port is associated with an Entity that accepts requests to detect changes in the internal state of the Entity and forwards messages or events when the states of interest become true.	Maps to an **interface** for requesting and managing change detection.

Table 9 Element Mappings

3.5 Mapping from the Relationship Profile

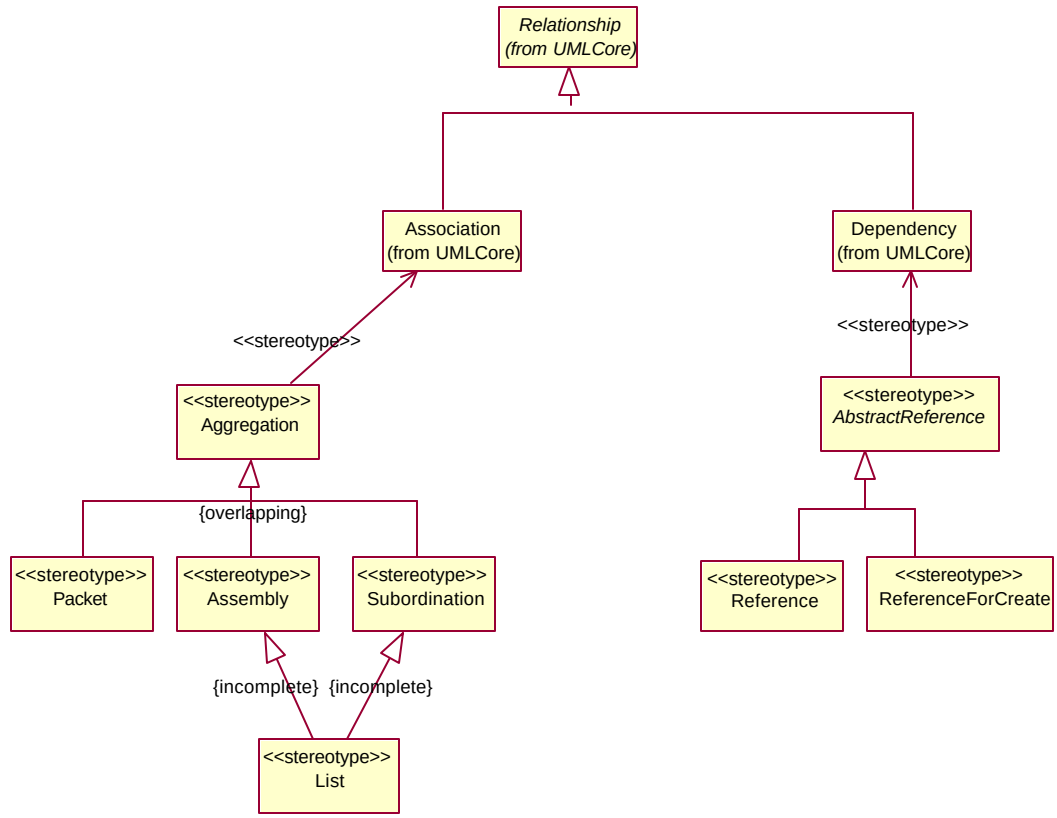


Figure 12 ECA Relationship model

The constraints specified for the different kinds of relationships can be mapped into code that executes to check and manage the constraints.

The relationship section of part I states that mapping algorithms should ignore all of the specific aggregation stereotypes defined in this profile that modify the a binary or non-binary aggregation (Assembly, Subordination, List, and Packet). These specific stereotypes are merely constraints on the multiplicities of the association ends. Any mapping of standard UML 1.4 aggregation associations would have to have rules for how the transformation is affected by these multiplicities.

The presence of the stereotypes does not mean that these multiplicities are missing. Therefore the multiplicities can drive the transformation and the stereotypes are redundant.

3.6 Mapping from the Event Profile

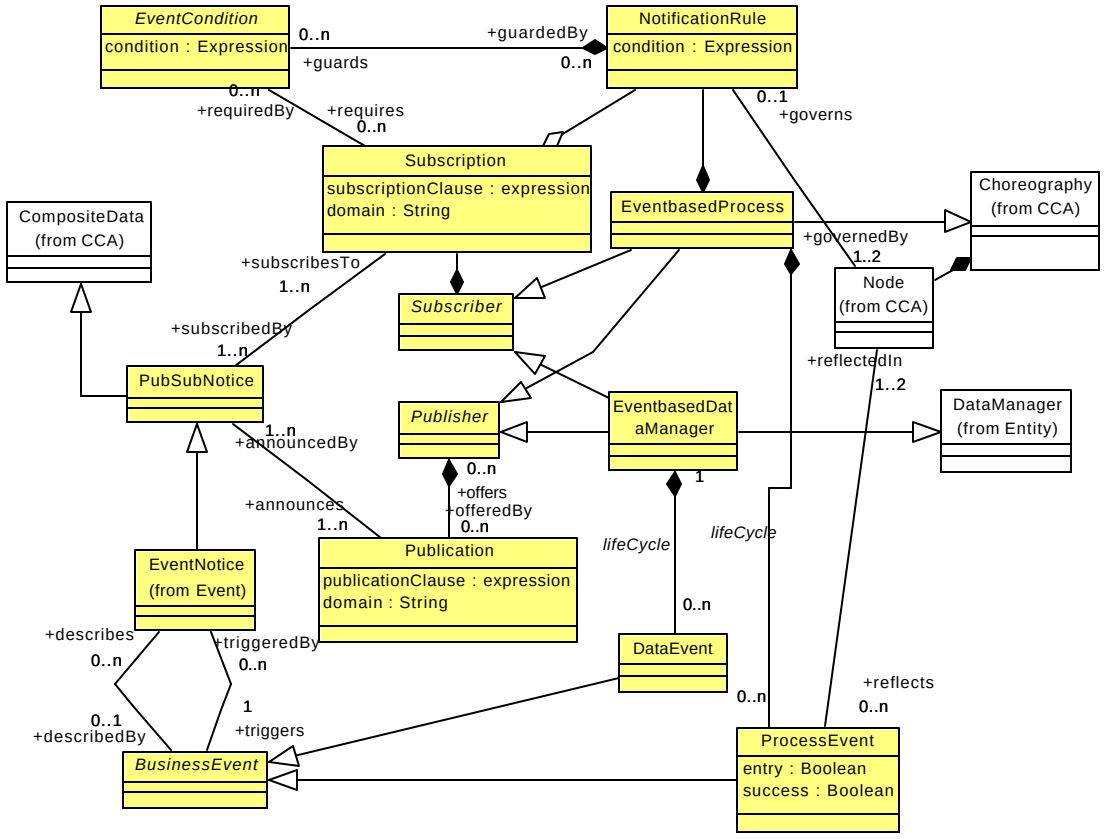


Figure 13 EDOC Event Model

Metamodel element	Mapping comment	EJB
Publisher	publisher is a component that exposes a list of publications, and produces PubSubNotices accordingly	*No inherent suppor for events. Can be mapped to specific event listener/provider interfaces for the listener/provider.*
Publication	Publication is a declaration of capability and intent to produce a PubSubNotice	*No inherent support in EJB.*

Metamodel element	**Mapping comment**	**EJB**
Subscriber	subscriber is a role or component that exposes a list of subscriptions, and consumes PubSubNotices accordingly	*No inherent support in EJB. Can be mapped to implementation of event listener interfaces.*
Subscription	Subscription is the expression of interest in receiving and capability to receive a PubSubNotice	*No inherent support in EJB. An EJB Bean can declare a subscription by implementing a event listener interface.*
PubSubNotice	PubSubNotice is any data structure that is *announcedBy* a publication and/or *subscribedTo* by a subscription. Instances of PubSubNotice are communicated as DataFlows from publishers to subscribers based on the subscriptions	*No inherent support. Can be mapped to an EventObject instance that is sent from an EJB provider to a listener.*
BusinessEvent	business event is any event of business interest that happens within an enterprise. BusinessEvents are either ProcessEvents or DataEvents	*No inherent support in EJB. Can be mapped to an instance of (a subtype of) EventObject.*
ProcessEvent	process event is any business event that reflects a state change within a process, i.e. entry into or exit from Nodes in a Choreography	*(same as above)*
DataEvent	data event is any business event that reflects a changes in data managed by a DataManager	*(same as above)*
EventNotice	event notice is any PubSubNotice that is triggered by a business event.	*No inherent support. Can be mapped to an EventObject instance that is sent from an EJB provider to a listener.*
EventBasedProcess	EventBasedProcess is a subtype of Choreography (CCA profile). It is a Subscriber and has NotificationRules associated with its Subscriptions. It is a Publisher and publishes ProcessEvents. ProcessEvents describe the life cycle of the EventBasedProcess	*No inherent support in EJB. Can be mapped to an EJB Bean that publishes java events.*
EventBasedDataManager	EventBasedDataManager is a DataManager. It is also a Publisher and publishes DataEvents when its data changes. It may also be a subscriber, typically subscribing to PubSubNotices relating to the maintenance of its data	*No inherent support in EJB.*
NotificationRule	NotificationRule is a rule associated with a subscription which determines what should happen within the EventBasedProcess holding the subscription when a qualifying PubSubNotice is delivered	*No inherent support in EJB.*

Metamodel element	**Mapping comment**	**EJB**
EventCondition	EventCondition identifies a subscription and specifies a PubSubNotice instance subset of which one must have been received to satisfy this condition	*No inherent support in EJB.*

Table 10 Mapping Events Concepts to Profile Elements

Event Publication and Event Subscription is mapped into Publication and Subscription as supported by the platform, or by event-notification in different messaging services.

An Event Notice is composite data that is being submitted through a flow port. It is also possible to map these to static callback interfaces.

3.7 Mapping from the Business Process Profile

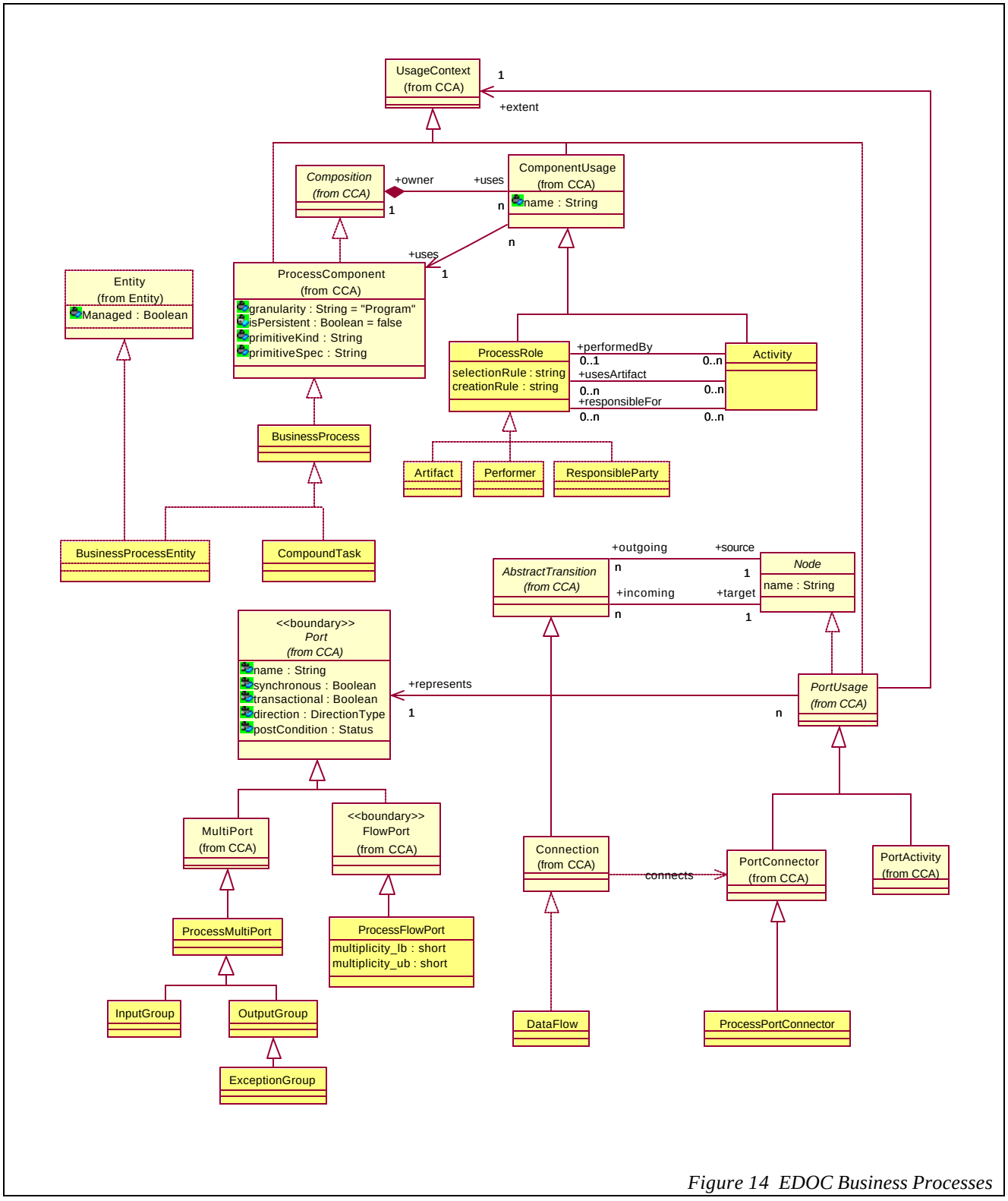


Figure 14 EDOC Business Processes

This model is organized with three main model elements to describe a business process: **BusinessProcess**, **CompoundTask** and **Activity** as shown in Figure 14 where the derivation from the CCA is shown. BusinessProccess is the outermost layer of composition representing a complete process specification. It is a ProcessComponent for the purpose of its usage inside other CCA Compositions, but its Composition is constrainedin the same way as a CompoubdTask.In other words, BusinessProcesses are the entry point from CCAto a process definition. CompoundTasks arealsospecializations of CCA ProcessComponents, but their Ports are constrained specializations of CCA Ports whichrepresent the data required to initiate an enactment itsComposition,which defines how itexecutes. The only ComponentUsages CompoundTasks and BusinessProcesses may contain are Activities, whichare specializations of CCA ComponentUsages. Activities are the pieces of work required to complete a Process, and CompoundTasks are the containers for a logical set of Activities and the **DataFlows** that define the temporal and data dependencies between them. DataFlows are specializations of CCA Flows that connect the PortConnectors on the Activities. Activities are always usages of a CompoundTask definition, which defines the Port types and their correlation semantics. CompoundTasks defining an Activity either compose additional Activities and DataFlows to show how this Activity is performed, or the Activity also refers to a **Performer ProcessRole** via the **performedBy** association, which is a binding to aProcessComponent that fulfils the requirements of the **ProcessRole**. Performer ProcessRoles are the exit point from a process defintion which allows it to invoke ProcessComponents (and their specializations, such as Entities). Many Activities may be usages of the same CompoundTask definition, and many activities in the same CompoundTask may be performed by the same **ProcessRole**.

Process models capture information at a level of abstraction which is complimentary to the information captured in Capsule/Port models such as CCA. Capsule/Port models define component composition and collaboration – the configuration/wiring of signals, data, and interactions between components. The Process profile, using information that describes enterprise processes, specifies *what* to wire and compose from the enterprise perspective.

The process models describes a higher level usage model for components. This information can be used by a workflow engine for sequencing the use of components. Mapping of the process profile to a technology must utilise available services on the target platform, e.g. CORBA workflow management facility on CORBA/CCM, IBMs web services flow language or possibly forthcoming process/workflow support for the J2EE-platform. The mapping below gives a possible mapping to the CCM and CORBA Workflow Management Facility.

Metamodel element name	**Mapping Comment**	**Mapping to a CORBA-like Workflow Management Facility**
CompoundTask	CompoundTasks have only a type nature.	WorkflowModel : WfProcessMgr and WorkflowModel : WfProcessObject
Activity	Activity	WorkflowModel : WfActivity
Activity : : usesArtifact	The usesArtifact association between Activity and ProcessRole is a way of defining access requirements of Activities to entities residing outside the process model.	Each link of this association kind is mapped as the existence of a NameValue member of the process_context attribute of the WfExecutionObject which implements this Activity
Activity::performedBy	The performedBy association specifies the ProcessRole which represents the behaviour to be executed by this Activity.	The association is implemented as a WorkflowModel::WfAssignment. The ProcessRole with which it is associated must support the type WorkflowModel::WfResource.

Metamodel element name	**Mapping Comment**	**Mapping to a CORBA-like Workflow Management Facility**
Activity::responsibleFor	The responsibleFor association between Activity and ProcessRole is a way of defining a party, represented by an object, which is responsible for the actions undertaken by this Activity.	Each link of this association kind is mapped as the existence of a NameValue member of the process_context attribute of the WfExecutionObject which implements this Activity
ProcessRole	ProcessRole	A set of CORBA object references in use from the context of a CORBA object.
BusinessProcess	A BusinessProcess is the implementation of a root CompoundTask in a tree of composed CompoundTask usages	CCM component
BusinessProcessEntity	BusinessProcessEntity	CORBAEntity
ProcessFlowPort	ProcessFlowPort	Maps to the creation and transmission of an event from an CCM event publisher/emitter.
ProcessPortConnector	ProcessPortConnector	Mapped to the representation of a logical link between an event sink and event publisher/emitter.
DataFlow	DataFlow	Dataflows of type source is mapped to CCM event publishers/emitters. Dataflows of types sink of mapped to CCM event sinks.
ProcessMultiPort	ProcessMultiPort	Maps to a CCM emitter of events.
InputGroup	InputGroup	Maps to
OutputGroup	OutputGroup	
ExceptionGroup	ExceptionGroup	
Performer	Performer	Maps to a CCM component that excetutes responsible a task.
Artifact	Artifact	Maps to a CCM component (e.g. CORBAEntity) representing the artifact.
ResponsibleParty	ResponsibleParty	Map to a CCM component responsible for the task.

Table 11 Mapping of process profile

3.8 Mapping from the Patterns Profile

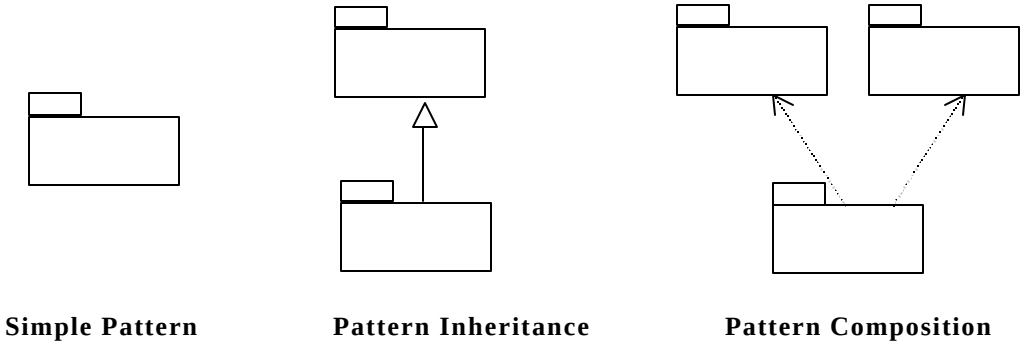


Figure 15 Pattern structures

The Patterns profile describes and examplifies the use of patterns fo rmodel specification. These are used in the other modeling profiles, but the patterns are typically unfolded before mapping to the platform specific models.

4. Mapping from EDOC to CORBA/CCM

This section describes a non-normative mapping from EDOC to the CORBA Component Model (CCM).

The mapping from EDOC to CCM is based on the same principles as the mapping from EDOC to EJB, as CCM can be considered a superset of EJB.

4.1 The Model of CORBA 3

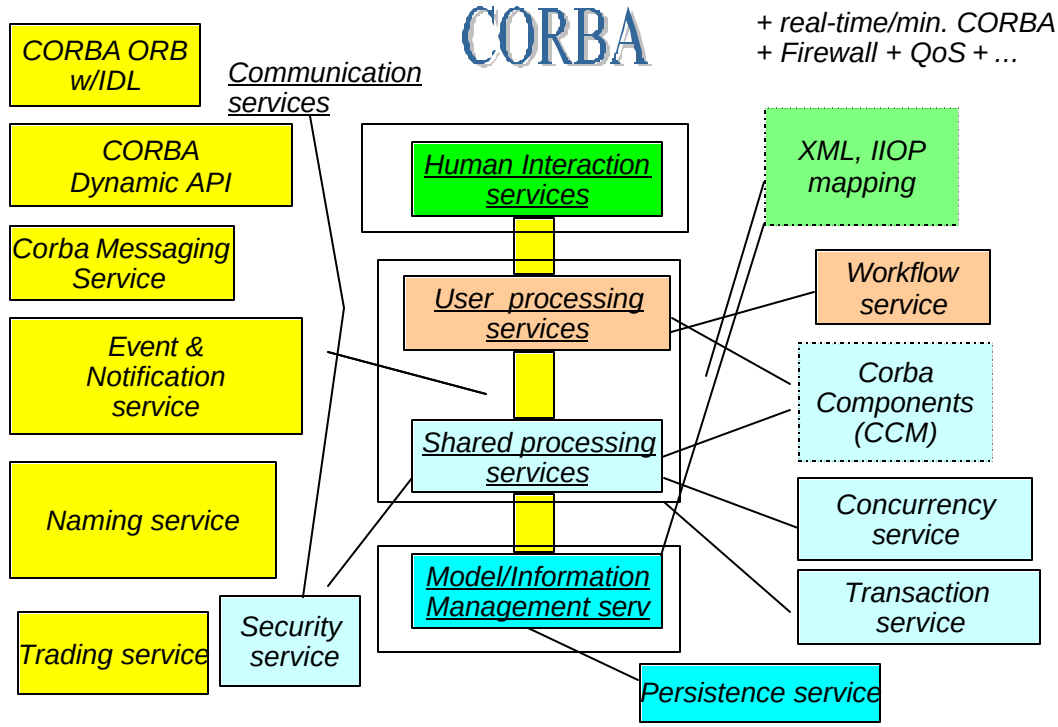


Figure 16 — CORBA Technologies related to Abstract Architecture model

Human Interaction Services are not directly supported.

Communication Services are supported by the CORBA ORB and dynamic API, as well as with the CORBA messaging service and event&notification service. Further support for communication of XML-structures will be provided by the CORBA XML-value mapping.

Workflow/Task Services are supported by the CORBA workflow service.

System Management Services are supported by CORBA Security and associated user services.

Processing Services are supported by server-side CORBA-objects and the concurrency and transaction service. In CORBA 2 the Corba Components Model will provide further services for server-side objects.

Model/Information Management Services are supported by the CORBA persistence service

4.2 CCM – The CORBA Component Model

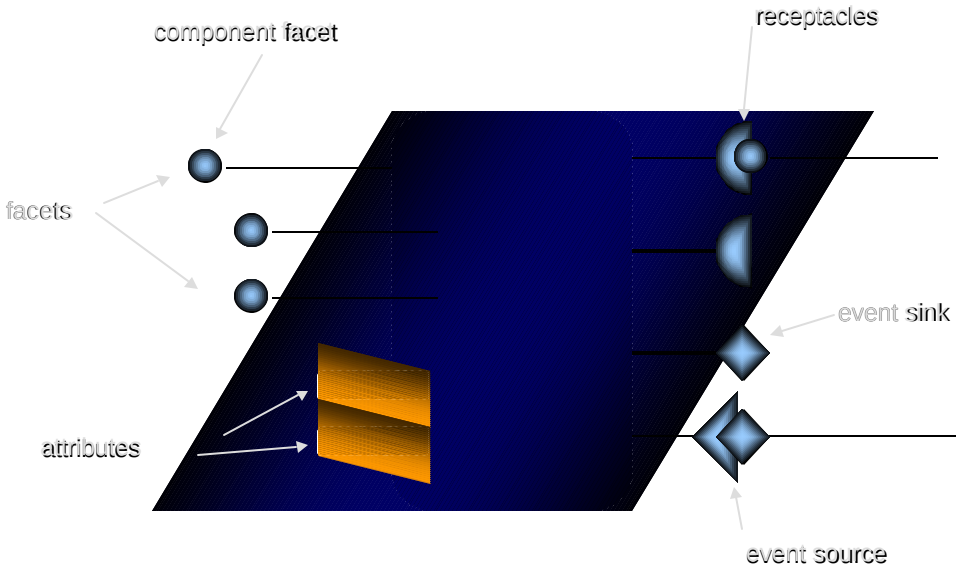


Figure 17 CCM External view

The CORBA Component Model (CCM) extends from the J2EE/EJB concepts, by describing a component with multiple outgoing (facets) and incoming (receptacles) interfaces, outgoing (source) and incoming (sink) event, and attributes for configuration.

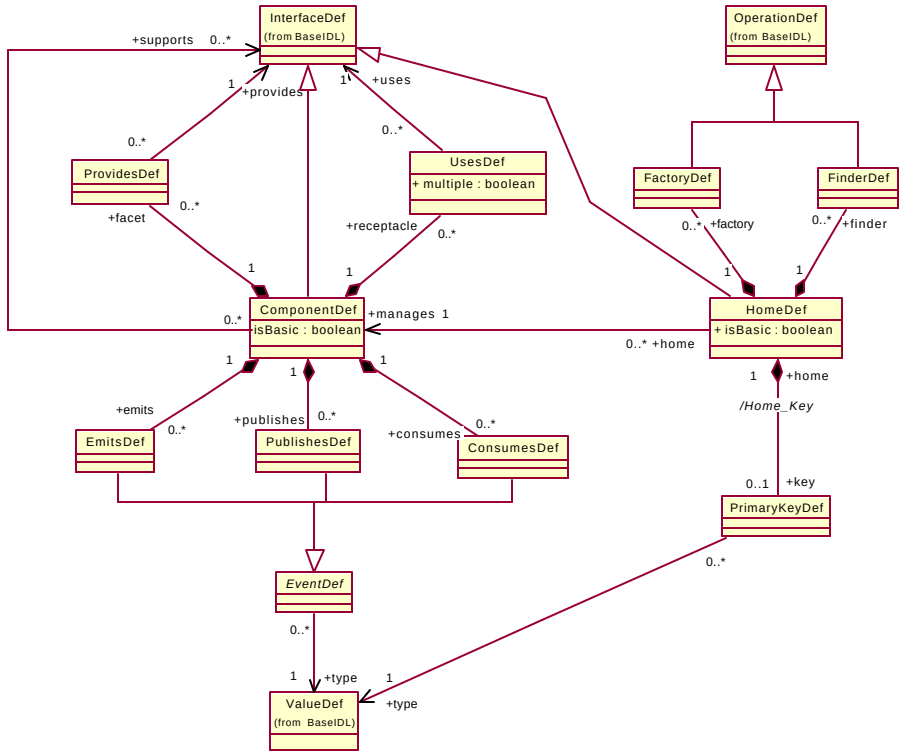


Figure 18 Corba Component Model

The UML diagram in the figure above illustrates the main concepts used for defining a CORBA Component.

4.3 UML Profile for CCM

There is currently now formalised UML Profile for CCM, although an initial draft has been done, in principal similar to the UML Profile for EJB.

4.3.1 Some suggested Stereotypes

Stereotype	Applies To	Definition
<<CORBAService>>	Subsystem (Design)	Indicates the Subsystem represents a CORBA service component.
<<CORBASession>>	Subsystem (Design)	Indicates the Subsystem represents a CORBA session component.
<<CORBAProcess>>	Subsystem (Design)	Indicates the Subsystem represents a CORBA process component.
<<CORBAEntity>>	Subsystem (Design)	Indicates the Subsystem represents a CORBA service component.
<<CORBAHome>>	Class (Design)	Indicates the class represents a CORBA home interface

Stereotype	Applies To	Definition
<<CORBAFinder>>	Class (Design)	Indicates the class represents a CORBA finder interface
<<CORBAEvent>>	Operation	Indicates the Operation represents a CORBA Event
<<CORBAConstant>>	Class (Design)	Indicates that the class represents a CORBA constant.
<<CORBAEnum>>	Class (Design)	Indicates that the class represents a CORBA Enum.
<<CORBAException>>	Class (Design)	Indicates that the class represents a CORBA Exception.
<<CORBAModule>>	Package (Design)	Indicates a package is a CORBA Module, as opposed to a logical abstraction.
<<CORBANative>>	Class (Design)	Indicates that the class represents a CORBA Native.
<<CORBAStruct>>	Class (Design)	Indicates that the class represents a CORBA Struct.
<<CORBATypedef>>	Class (Design)	Indicates that the class represents a CORBA Typedef.
<<CORBAUnion>>	Class (Design)	Indicates that the class represents a CORBA Union.
<<CORBAFacet>>	Component (Design)	Indicates the component represents a CORBA facet

Table 12: UML Profile for CCM – Suggested Prototypes

4.3.2 Tagged Values

Tagged Value	Applies To	Definition
ContainerType	Subsystem <<CORBAService>> <<CORBASession>> <<CORBAProcess>> <<CORBAEntity>>	Usage is: Transient Persistent
ContainerImplementation Type	Subsystem <<CORBAService>> <<CORBASession>> <<CORBAProcess>> <<CORBAEntity>>	Usage is: Stateless Conversational Durable
ServantLifetimePolicy	Subsystem <<CORBAService>> <<CORBASession>> <<CORBAProcess>> <<CORBAEntity>>	Usage is: Method Transaction Component Container
TransactionPolicy	Subsystem <<CORBAService>> <<CORBASession>> <<CORBAProcess>> <<CORBAEntity>>	Usage is: NOT_SUPPORTED REQUIRED SUPPORTS REQUIRES_NEW MANDATORY NEVER

Tagged Value	**Applies To**	**Definition**
SecurityPolicy	Subsystem <<CORBAService>> <<CORBASession>> <<CORBAProcess>> <<CORBAEntity>>	TBD
EventPolicy	Subsystem <<CORBAService>> <<CORBASession>> <<CORBAProcess>> <<CORBAEntity>>	Usage is: Normal Default Transaction
PersistenceMechanism	Subsystem <<CORBAService>> <<CORBASession>> <<CORBAProcess>> <<CORBAEntity>>	Usage is: CORBA User
PersistenceSupport	Subsystem <<CORBAService>> <<CORBASession>> <<CORBAProcess>> <<CORBAEntity>>	Usage is: Container Managed Component Managed
ImplementationType	Class <<CORBAConstant>> Class <<CORBATypedef>> Class <<CORBASequence>> Class <<CORBAUnion>>	Usage is dependent on the class stereotype: CORBAConstant – the type of the constant CORBATypedef or CORBASequence – the type of the typedef or sequence if there is no dependency relationship CORBAUnion – the switch type
ConstValue	Class <<CORBAConstant>>	Used only if the stereotype of the class is CORBAConstant. Represents the value of the constant.
ArrayDimensions	Class <<CORBATypedef>>	If non-blank, indicates that the *declarator* is an array and defines the array dimension(s) portion of the *declarator.*
CaseSpecifier	Attribute <<CORBAUnion>> Role <<CORBAUnion>>	Used only when the stereotype of the class is CORBAUnion. Case expression. Should be equal to 'default' for the default case.
IsReadOnly	Attribute Role	Indicates whether or not the attribute is readonly. BOOLEAN = FALSE

Tagged Value	Applies To	Definition
Order	Attribute	Integer which defines the ordering of the attributes.
ArrayDimensions	Attribute <<CORBAException>> <<CORBAStruct>> <<CORBAUnion>>	Used only if the attribute's class represents an exception, struct, or union. If non-blank, indicates that the *declarator* is an array and defines the array dimension(s) portion of the *declarator.*
BoundedRoleType	Role	Whether to use an array or sequence to represent a relationship with bounded cardinality. Unbounded cardinality always generates an unbounded sequence. The cardinality of the relationship defines the size of the array or sequence. ENUMERATION (Sequence, Array) = Sequence

Table 13: UML Profile for CCM – Tagged Values

4.4 Mapping from the EDOC CCA Profile

This section details the mappings from the EDOC Part I CCA, Component Collaboration Architecture.

4.4.1 Mapping Process Components and Protocols

Part of a component's specification is the set of **protocols** it implements, a protocol specifies what messages the component sends and receives when it collaborates with another component and the **choreography** of those messages – when they can be sent and received. Each protocol the component supports is provided via a "**port**", the connection point between components.

Metamodel element name	Map comment	CCM
ProcessComponent	A ProcessComponent represents the contract for a component that performs actions – it "does something". A ProcessComponent may realize a set of Ports for interaction with other ProcessComponents and it may be configured with properties	Maps to a **<<CORBA Component>>** (in this description) but can also naturally map to higher (Business Process) or lower (Object) level concepts. (One of CORBAService, COR BASession, CORBAProcess or CORBAEntity)

Metamodel element name	**Map comment**	**CCM**
IsPersistent (Property of ProcessComponent)	default=false, if true stores session specific state across interactions	If true, Component = CORBASession. If false Component = CORBAService (default). Is always true for CORBAProcess and CORBAEntity
Port	A port realizes a simple or complex conversation for a ProcessComponent or protocol. Each port is connected with collaborative components that speak the same protocol All interactions with a ProcessComponent are done via one of its ports. Each port provides a connection point for interaction with other components or services and realizes a specific protocol. The protocol may be simple and use a "FlowPort" or the protocol may be complex and use a "ProtocolPort" or an "OperationPort". If allowed by its protocol, a port may send and receive information.	Mapping depends on kind of port. See below. **CORBAFacet** and **CORBAReceptacle** or **EventSource** or **EventSink**
IsTransactional (Property of port)	interactions with the component are transactional & atomic	IsTransacational = true means a mapping to an CCMTransactionSupportKind of one of supported, required, requiresNew, mandatory, selfManaged IsTransacational = false means a mapping to CCMTransactionSupportKind of one of notSupported or Never
IsSynchronous (Property of port)	A port may interact synchronously or asynchronously. A port that is marked as synchronous is required to interact using synchronous messages and return values	IsSynchronous = true means mapping to operations, IsSynchronous = false (default) means mapping to a one-way operation if this is an initiator port or to event source or sink
Direction (Property of port) Initiates or Responds	Indicates that the port will either initiate or respond to the related type. An initiating port will send the first message. Note that by using ProtocolPorts a port may be the initiator of some protocols and the responder to others.	The direction of the port is with respect to the protocol, it is only implicitly mapped to CCM (See below)

Metamodel element name	Map comment	CCM
FlowPort	A Flow Port is a port which defines a data flow in or out of the port on behalf of the owning component or protocol. direction in is one way operation or out is a one way call	Direction *Initiates*: Maps to a **one-way operation being invoked,** or to a **CCM event source**. Direction *Responds*: Maps to a **message being received**. In the synchrounous case this is a mapping to an operation without return values in an interface, In the asynchrounous case this is a mapping to an CCM event sink.
ProtocolPort	A protocol port is used for potentially complex two-way interactions between components	Maps to the **CORBAReceptacle** and/or **CCM Event sink** being used and/or provided for a two-way interaction
MultiPort	Each port owned by the MultiPort will "buffer" information sent to that port until all the ports within the MultiPort have received data, at this time all the ports will send their data	Direction *Responds*: Maps to an implementation where a set of messages (events) need to be received by the port before it is sent further.
OperationPort	An operation port represents the typical call/return pattern of an operation. The OperationPort is a PortOwner which is constrained to contain only flow ports, exactly one of which must have its direction set to "initiates".	Direction *Initiates*: This maps to the **operation to be invoked**, in a CORBAReceptacle. Direction *Responds*: This maps to an **operation to be implemented** in a CORBAFacet.

Metamodel element name	Map comment	CCM
Protocol	A protocol defines a type of conversation between two parties, the initiator and responder. One protocol role is the initiator of the conversation and the other the responder. However, after the conversation has been initiated, individual messages and sub-protocols may by initiated by either party.	Maps to a **description of the messages and operation interactions in a conversation**. This can be described through the operations in receptacles and facets that are involved, and the event sources and sinks that are involved.
Interface	An interface is a protocol constrained to match the capabilities of the typical object interface. It is constrained to only contain OperationPorts and FlowPorts and all of its ports must respond to the interaction (making interfaces one-way) Each OperationPort or FlowPort in the Interface will map to a method. A ProtocolPort which initiates the Interface will call the interface. A ProtocolPort which Responds will implement the interface Existing interface	An EDOC interface represents a protocol that maps directly to a UML interface that again is mapped to corresponding CCM facet and receptacle.
InitiatingRole	The role of the protocol which will send the first message Default Interface name	Represents the **CCM component** that is the initiator of a protocol
RespondingRole	The role in the protocol which will receive the first message Default Interface name	Represents the **CCM component** that is the receiver of the first message in a protocol
PropertyDefinition	PropertyDefinition defines name and type for properties which may be set when the ProcessComponent is used	Properties on the CCM component.

Table 14: Stereotypes for Structural Specification (UML notation: Class Diagram)

A protocol specifies the conversation between two ProcessComponents (via their ports). Each component that is using that protocol must use it from the perspective of the "initiating role" or the "responding role". Each of these components will use every port in the protocol, but in complementary directions.

Each port is connected with collaborative components that speak the same protocol. Multi-party conversions are defined by components using multiple ports, one for each kind of party.

Components interact with their environment through ports. A port has a defined interaction protocol. Ports may send messages, receive messages, or both. A port may be implemented as an object interface, e.g., CORBA or Java interface.

Ports are synchronous or asynchronous. A synchronous port communicates within the context of a transaction. An asynchronous port communicates in a store-and-forward

manner so that sending a message occurs in the context of one transaction and receipt of the message then occurs in the context of another transaction.

Ports may communicate with messages or event notices. A message is directed to a specific destination. An event notice is published to the communication infrastructure to be delivered to subscribers—destinations that have expressed interest. The messages and event notices may be communicated synchronously or asynchronously.

All Data Managers will have a synchronous interface port that represents the typical object interface. A Data Manager may have other ports, such as to send messages to other Data Managers, and to send and receive asynchronous messages and events.

4.4.2 Mapping Composition

Components may be abstract (having only an outside) or concrete (having an inside and outside). Frequently a concrete component inherits its external contract from an abstract component – implementing that component.

There may be any number of implementations for an **abstract component** and various ways to "bind" the correct implementation when a component is used.

The two basic kinds of concrete components are:

- **Primitive components** – those that are built with programming languages or by wrapping legacy systems.

- **Composed components** – Components that are built from other components; these use other components to implement the new components functionality. Composed components are defined using a **composition**.

Compositions define how components are *used*. Inside of a composition components are used, configured and connected. This connected set of component usage's implements the behavior of the composition in terms of these other components – which may be primitive, composed or abstract components.

Compositions can also include a **choreography** of how the components used work together, which should execute when.

Compositions are used to build composed components out of other components and to describe community processes – how a set of large grain components works together for some purpose.

Central to compositions are the **connections** between components, values for **configuration properties** and the ability to **bind** concrete components to a component usage.

Metamodel element name	Mapping description	CCM
Composition	Compositions describe how instances of ProcessComponents (called ComponentUsages) are configured (with PropertyValues and ContextualBindings) and connected (with Connections) to implement the composed ProcessComponent or CommunityProcess.	Maps to the **structure of interacting components** and how they are configured and connected. Through the configuration and connection between receptacles and facets, and event sources and sinks.
ComponentUsage	A ComponentUsage will cause a ProcessComponent instance to be created at runtime (this instantiation may be real or virtual).	Maps to **the use of one component** by another.
PortConnector	PortConnector provides a "connection point" for ComponentUsages within a composition and exposes the defined ports within the composition. The connections between PortConnectors are made with Connections.	Maps to the realisation of ports. The runtime representation of **CORBA Receptacle** represents initiating ports. For receiving ports this is the representation of a **CORBA facet**. The mapping can also be done to **CCM event publisher and emitters**.
Connection	A Connection connects two PortConnectors within a composition. Each port can produce and/or consume message events. The connection logically registers each port connector as a listener to the other, effectively making them collaborators.	This is the representation of the communication between two PortConnectors, either the receptacle/facet link or the event publisher/emitter link.
PropertyValue	a ProcessComponent may have configuration properties –which are defined by a PropertyDefinition. When the component is used in a ComponentUsage those properties values may be set using a PropertyValue.	**The value of CORBA Component attributes.**
ContextualBinding	Contextual Binding allows the substitution of a more concrete ProcessComponent for a compatible abstract ProcessComponent when an abstract composed ProcessComponent is used.	Interface conformance allows for **multiple implementations** of a CORBA Component.
CommunityProcess	CommunityProcess may be thought of as the "top level composition" in a CCA specification, it is a specification of a composition of ProcessComponents that work together for some purpose other than specifying another ProcessComponent.	**Highest level of component interaction**

Table 15: Stereotypes for Composition (UML notation: Collaboration Diagram at specification level)

Specifications of composition can be used to automatically create components that use existing components, and support this dynamically or through code generation.

4.4.3 Mapping Choreography

A Choreography specifies how messages will flow between PortUsages. The choreography may be externally oriented, specifying the contract a component will have with other components or, it may be internally oriented, specifying the flow of messages within a composition. External choreographies are shown as an activity graph while internal choreography is shown as part of a collaboration. An external choreography may be defined for a protocol or a ProcessComponent.

A **Choreography** uses **Connections** and **transitions** to order port messages as a state machine. Each "node" in the choreography must refer to a state or a port usage.

Choreography is an abstract capability that is inherited by ProcessComponents and protocols.

Choreography may be used at multiple levels;

- A protocol Choreography specifies the sequencing of messages and sub-protocols between protocol roles. This is much like a sequence diagram.

- A process component Choreography specifies the sequencing of multiple messages and sub-protocols of ports and is part of the external contract of the component.

The use of choreography at all of these levels is not always required, as sufficient specification may be determined from the other layers.

Specifications of choreography can be used to automatically create executable code, or to support the execution of a workflow engine, in the case of a mapping to a coordinating workflow system.

4.4.4 Mapping Document Model

The information that flows between components is described in a **Document Model**, the structure of information exchanged. The document model also forms the basis for information entities and a generic **information model**. The information model is acted on by CCA process components.

A **data element** represents a type of data which may either be primitive **DataTypes** or composite**. CompositeData** has named attributes which reference other types. Any type may have a **DataInvariant** expression.

Attributes may be **isByValue**, which are strongly contained or may simply reference other data elements provided by some external service. Attributes may also be marked as **required** and/or **many** to indicate cardinality. **DataTypes** define local data – these types are defined outside of CCA. **ExternalDocument** defines a document defined in an external type system. An **enumeration** defines a type with a fixed set of values

Metamodel element name	Mapping Comment	CCM
CompositeData	A datatype composed of other types in the form of attributes	Maps to a **data representation** as being used for a set of actual arguments in a message/operation or in a generic data structure
ExternalDocument	A large, self contained document defined in an external type system such as XML, Cobol or Java that may or may not map to the ECA document model	Mapping to an external datastructure to be interpreted.
DataInvariant	A constraint on the legal values of a data element.	Constraint that could be checked by generated code.
DataType	Data types may have their structure and semantics defined outside of CCA. The following data types are defined for all specializations of CCA: String, Integer, Float, Decimal, Boolean.	Maps to the **basic datatypes of CORBA IDL.**
Enumeration	An enumeration defines a type that may have a fixed set of values.	Maps to CORBA IDL enumeration.
Attribute	Defines one "slot" of a composite type that may be filled by a data element of "type".	Maps to the definition of **one attribute in CompositeData.**

Table 16: Stereotypes for DocumentModel (UML notation: Class Diagram)

Specifications of data elements from the document model are used for creating value objects for messages, and for defining data representations.

4.5 Mapping from the Entities Profile

Data Managers are components which manages a given type of data. **Entities** are identifiable elements in an information model which are managed by **Entity Managers**. **Keys** provide the identity for entities.

Metamodel Element	Mapping comment	CCM
Data Manager	Data Manager is a functional component that provides access to and may perform operations on its associated Composite Data (i.e., its state). The Data Manager defines ports for access to operations on the state data	Maps typically to an **CORBAEntity**, but might sometimes be handled by a CORBAProcess or a CORBASession. If not network-addressable (see below) this might be handled by a dependent object.
Network-addressable	A Boolean value which indicates if the Data Manager is intended to be accessible over the network.	If true, maps the DataManager to an **CORBASession, CORBAProcess or CORBAEntity**.
Shareable	Boolean value which indicates if the Data Manager can be shared by multiple transactions/sessions. A Data Manager that is not sharable is either transient or depends on a sharable Data Manager that contains it for persistence.	If true, maps the DataManager to a **CORBAProcess or CORBAEntity**.

Metamodel Element	Mapping comment	CCM
Entity	Entity is an object representing something in the real world of the application domain. It incorporates Entity Data that represents the state of the real world thing, and it provides the functionality to encapsulate the Entity Data and provide associated business logic.	Maps to an **CORBAProcess or CORBAEntity** (if *Shareable* and/or *Network-addressable, or Managed)* if not to **a dependent object**.
Managed (Entity Property)	Boolean value that indicates if the Entity type is *managed*. If it is managed, then the implementation provides a mechanism for accessing the extent of all instances	If true implies the declaration of an **CORBAFactoryInterface**.
Entity Data	Entity Data is the data structure that represents a concept in the business domain. It is equivalent to an entity in data modelling or a relation in a relational database. In a Data Manager or its specializations, such as Entity, it represents the state of an object.	Maps to the data representation part of a **CORBAEntity.**
Entity Role	Entity Role extends its parent Entity for participation in a particular context. An Entity may have a number of associated Entity Roles reflecting participation in multiple contexts	Maps to another associated CORBAEntity or dependent object
Virtual Entity	Boolean value that indicates if the Entity Role incorporates and extends the primary interface of the parent Entity it represents, i.e., it can be used in place of the primary Entity	Maps to another associated **CORBAEntity or dependent object**, extending the primary interface.
Key	Key is composed of key elements which may be selected attribute values of the associated Entity Data or Foreign Keys	Maps to a **CORBAPrimaryKey**.
Foreign Key	A Foreign Key is the key of a related Entity Data.	Maps to **CORBAPrimaryKey** for another CORBAEntity
Data Probe	Data Probe port is associated with an Entity that accepts requests to detect changes in the internal state of the Entity and forwards messages or events when the states of interest become true.	Maps to an **interface** for requesting and managing change detection.

Table 17 Element Mappings

4.6 Mapping from the Relationship Profile

The constraints specified for the different kinds of relationships can be mapped nto code that executes to check and manage the constraints.

The relationship section of part I states that mapping algorithms should ignore all of the specific aggregation stereotypes defined in this profile that modify the a binary or non-binary aggregation (Assembly, Subordination, List, and Packet). These specific stereotypes are merely constraints on the multiplicities of the association ends. Any mapping of standard UML 1.4 aggregation associations would have to have rules for how the transformation is affected by these multiplicities.

The presence of the stereotypes does not mean that these multiplicities are missing. Therefore the multiplicities can drive the transformation and the stereotypes are redundant.

4.7 Mapping from the Event Profile

Metamodel element	**Mapping comment**	**CCM**
Publisher	publisher is a component that exposes a list of publications, and produces PubSubNotices accordingly	Maps to a **CORBA component that publishes events** (CCM publisher/emitter)**.**
Publication	Publication is a declaration of capability and intent to produce a PubSubNotice	A publication is declared by **publishing/emitting CCM components**.
Subscriber	subscriber is a role or component that exposes a list of subscriptions, and consumes PubSubNotices accordingly	Mapped to **CCM event sink.**
Subscription	Subscription is the expression of interest in receiving and capability to receive a PubSubNotice	A subscription is declared by a **CCM event sink**.
PubSubNotice	PubSubNotice is any data structure that is *announcedBy* a publication and/or *subscribedTo* by a subscription. Instances of PubSubNotice are communicated as DataFlows from publishers to subscribers based on the subscriptions	The **notification of a CCM event** sent from a CCM event publisher/emitter.
BusinessEvent	business event is any event of business interest that happens within an enterprise. BusinessEvents are either ProcessEvents or DataEvents	**CCM event**.
ProcessEvent	process event is any business event that reflects a state change within a process, i.e. entry into or exit from Nodes in a Choreography	**CCM event**.
DataEvent	data event is any business event that reflects a changes in data managed by a DataManager	**CCM event**.
EventNotice	event notice is any PubSubNotice that is triggered by a business event.	The **notification of a CCM event** sent from a CCM event publisher/emitter.

Metamodel element	Mapping comment	CCM
EventBasedProcess	EventBasedProcess is a subtype of Choreography (CCA profile). It is a Subscriber and has NotificationRules associated with its Subscriptions. It is a Publisher and publishes ProcessEvents. ProcessEvents describe the life cycle of the EventBasedProcess	Maps to a **CCM component** that is a publisher/emitter of CCM process events.
EventBasedDataManager	EventBasedDataManager is a DataManager. It is also a Publisher and publishes DataEvents when its data changes. It may also be a subscriber, typically subscribing to PubSubNotices relating to the maintenance of its data	Maps to CCM components that is a Maps to **CCM event sink**.
NotificationRule	NotificationRule is a rule associated with a subscription which determines what should happen within the EventBasedProcess holding the subscription when a qualifying PubSubNotice is delivered	Maps to the **logic provided by a CCM event sink** for handling incoming notifications.
EventCondition	EventCondition identifies a subscription and specifies a PubSubNotice instance subset of which one must have been received to satisfy this condition	

Table 18 Mapping Events Concepts to Profile Elements

Event Publication and Event Subscription is mapped into Publication and Subscription as supported by the DCP, or by event-notification in different messaging services.

An Event Notice is composite data that is being submitted through a flow port. It is also possible to map these to static callback interfaces.

4.8 Mapping from the Business Process Profile

This mapping is described in more detail in Chapter 5.

This model is organized with three main model elements to describe a business process: **BusinessProcess**, **CompoundTask** and **Activity** as shown in Figure 14 where the derivation from the CCA is shown. BusinessProccess is the outermost layer of composition representing a complete process specification. It is a ProcessComponent for the purpose of its usage inside other CCA Compositions, but its Composition is constrainedin the same way as a CompoubdTask.In other words, BusinessProcesses are the entry point from CCAto a process definition. CompoundTasks arealsospecializations of CCA ProcessComponents, but their Ports are constrained specializations of CCA Ports whichrepresent the data required to initiate an enactment itsComposition,which defines how itexecutes. The only ComponentUsages CompoundTasks and BusinessProcesses may contain are Activities, whichare specializations of CCA ComponentUsages. Activities are the pieces of work required to complete a Process, and CompoundTasks are the containers for a logical set of Activities and the **DataFlows** that define the temporal and data dependencies between them. DataFlows are specializations of CCA Flows that connect the PortConnectors on the Activities. Activities are always usages of a CompoundTask definition, which defines the

Port types and their correlation semantics. CompoundTasks defining an Activity either compose additional Activities and DataFlows to show how this Activity is performed, or the Activity also refers to a **Performer ProcessRole** via the **performedBy** association, which is a binding to aProcessComponent that fulfils the requirements of the **ProcessRole**. Performer ProcessRoles are the exit point from a process defintion which allows it to invoke ProcessComponents (and their specializations, such as Entities). Many Activities may be usages of the same CompoundTask definition, and many activities in the same CompoundTask may be performed by the same **ProcessRole**.

Process models capture information at a level of abstraction which is complimentary to the information captured in Capsule/Port models such as CCA. Capsule/Port models define component composition and collaboration – the configuration/wiring of signals, data, and interactions between components. The Process profile, using information that describes enterprise processes, specifies *what* to wire and compose from the enterprise perspective.

The process models describes a higher level usage model for components. This information can be used by a workflow engine for sequencing the use of components. Mapping of the process profile to a technology must utilise available services on the target platform, e.g. CORBA workflow management facility on CORBA/CCM, IBMs web services flow language or possibly forthcoming process/workflow support for the J2EE-platform. The mapping below gives a possible mapping to the CCM and CORBA Workflow Management Facility.

Metamodel element name	**Mapping Comment**	**CCM/ CORBA Workflow Management Facility**
CompoundTask	CompoundTasks have only a type nature.	WorkflowModel : WfProcessMgr and WorkflowModel : WfProcessObject
Activity	Activity	WorkflowModel : WfActivity
Activity : : usesArtifact	The usesArtifact association between Activity and ProcessRole is a way of defining access requirements of Activities to entities residing outside the process model.	Each link of this association kind is mapped as the existence of a NameValue member of the process_context attribute of the WfExecutionObject which implements this Activity
Activity::performedBy	The performedBy association specifies the ProcessRole which represents the behaviour to be executed by this Activity.	The association is implemented as a WorkflowModel::WfAssignment. The ProcessRole with which it is associated must support the type WorkflowModel::WfResource.
Activity::responsibleFor	The responsibleFor association between Activity and ProcessRole is a way of defining a party, represented by an object, which is responsible for the actions undertaken by this Activity.	Each link of this association kind is mapped as the existence of a NameValue member of the process_context attribute of the WfExecutionObject which implements this Activity
ProcessRole	ProcessRole	A set of CORBA object references in use from the context of a CORBA object.
BusinessProcess	A BusinessProcess is the implementation of a root CompoundTask in a tree of composed CompoundTask usages	CCM component
BusinessProcessEntity	BusinessProcessEntity	CORBAEntity

Metamodel element name	Mapping Comment	CCM/ CORBA Workflow Management Facility
ProcessFlowPort	ProcessFlowPort	Maps to the creation and transmission of an event from an CCM event publisher/emitter.
ProcessPortConnector	ProcessPortConnector	Mapped to the representation of a logical link between an event sink and event publisher/emitter.
DataFlow	DataFlow	Dataflows of type source is mapped to CCM event publishers/emitters. Dataflows of types sink of mapped to CCM event sinks.
ProcessMultiPort	ProcessMultiPort	Maps to a CCM emitter of events.
InputGroup	InputGroup	Maps to
OutputGroup	OutputGroup	
ExceptionGroup	ExceptionGroup	
Performer	Performer	Maps to a CCM component that excetutes responsible a task.
Artifact	Artifact	Maps to a CCM component (e.g. CORBAEntity) representing the artifact.
ResponsibleParty	ResponsibleParty	Map to a CCM component responsible for the task.

Table 19 Mapping of process profile

4.9 Mapping from the Patterns Profile

The Patterns profile describes and examplifies the use of patterns fo rmodel specification. These are used in the other modeling profiles, but the patterns are typically unfolded before mapping to the platform specific models.

5. Mapping From EDOC Business Process to CORBA

5.1 Common Base Types for the Business Process Model

The Workflow Management Facility defines a number of interfaces for the execution, monitoring and meta-data query of what we have modeled as Activities, CompoundTasks and Business Processes. These are used as a common basis for the alternative mappings of the Business Process Model.

5.1.1 BusinessProcess

A BusinessProcess is the implementation of a root CompoundTask in a tree of composed CompoundTask usages, and as such, it is implemented by a WorkflowModel::WfProcessMgr object, as defined in the mapping of CompoundTask.

5.1.2 CompoundTask

CompoundTasks have only a type nature. CompoundTask is therefore mapped to a type manager, which in the Workflow Management facility are WorkflowModel::WfProcessMgr objects.

5.1.3 Activity

Activities are mapped to WorkflowModel::WfActivity objects. Through the mapping of the Activity's InputGroups and usesArtifact and repsonsibleFor associations, it will be able to pass Input values and references to bound entities.

During execution the enabling of an Activity whose ports and their contracts are defined by a CompoundTask causes an Activity instance to be created. The external contract nature of this instance is mapped as a WorkflowModel::WfProcess object. The key attribute of the WfProcess must be an instance identifier. This identifier is used (as a parent Task Id) in the mappings of the DataGroups and DataFlows in the following sections.

When the WfProcess implementing the Activity is run, it must also create instances of WorkflowModel::WfActivity for each Activity that is defined by its CompoundTask's Composition.

The complete mapping of an Activity depends on whether it has a performedBy association to a ProcessRole or whether its execution is defined by the Composition of its CompoundTask definition; Activities that represent a Composition are mapped to objects that also implement the WorkflowModel::WfRequester interface.

Associations

usesArtifact

The usesArtifact association between Activity and ProcessRole is a way of defining access requirements of Activities to entities residing outside the process model. Each link of this association kind is mapped as the existence of a NameValue member of the process_context attribute of the WfExecutionObject which implements this Activity; the_name in the NameValue is given the ProcessRole's name, and the_value is an object reference of the same type as the ProcessRole. At runtime the object referred to will be chosen (using the type association and the SelectionRule or CreationRule Expressions). See the mapping of ProcessRole in Section 5.1.4 for details.

performedBy

The performedBy association specifies the ProcessRole which represents the behaviour to be executed by this Activity. The nominated ProcessRole may represent the interface to a person or group of people, or it may be a fully automated program that processes the Activity's inputs and produces some outputs. The association is implemented as a WorkflowModel::WfAssignment. The ProcessRole with which it is associated must support the type WorkflowModel::WfResource.

The WfResource's resource_key and resource_name attributes may be used by the ProcessRole to locate and bind an entity of the appropriate application type to perform the Activity. Often this entity will represent a work list that will use the Activity's name, inputs, and the ProcessRoles participating in usesAtrifact and

repsonsibleFor associations with this Activity, to create a work item which is sent to a person, or group of people for processing.

repsonsibleFor

The repsonsibleFor association between Activity and ProcessRole is a way of defining a party, represented by an object, which is responsible for the actions undertaken by this Activity. Each link of this association kind is mapped as the existence of a NameValue member of the process_context attribute of the WfExecutionObject which implements this Activity; the_name in the NameValue is given the ProcessRole's name, and the_value is an object reference of the same type as the ProcessRole. At runtime the object referred to will be chosen (using the type association and the SelectionRule or CreationRule Expressions). See the mapping of ProcessRole in Section 5.1.4 for details.

5.1.4 ProcessRole

A ProcessRole is mapped in CORBA as a set of object reference variables in use in some context. This is a novel modeling concept in the OMA, as specifications of clients of CORBA objects, and the binding process by which client code comes to refer to the "right" objects, has been impossible until now.

The ProcessRole concept recognizes that interface type compatibility is not sufficient to ensure that an object implementing the correct behavioral semantics is invoked by a client. ProcessRole is a kind of abstract behavior, with both an interface type slot, and two kinds of criteria for selection of object instances to fill the role:

- Its SelectionRule attribute - which allows the behaviour specification to express criteria by which objects that may fill the ProcessRole may be selected.
- Its CreationRule attribute - which allows creation of objects which may then fill the ProcessRole.

5.1.4.1 Binding

The mapping for filling a ProcessRole is as follows. For each ProcessRole, an instance of a business entity (a CORBA Object) must be located. The model elements provide a number of options to modelers to specify their binding constraints. Here are some of them:

The SelectionRule expression of the ProcessRole may provide:

- a key for use with a factory/finder (type manager) in order to locate an appropriate object;
- an Interoperable Naming iiopname or iioploc URL which nominates a specific object;
- a Naming Context or hierarchy of Contexts which contain appropriate objects;
- a Trader Service Type and Constraint expression which will match appropriate objects in the Business Domain's Trader.

The CreationRule expression of the ProcessRole may provide:

- a key for use with a factory/finder (type manager) in order to create an appropriate object;

- appropriate parameters for passing to a Factory to construct a new object.

All of these options are available as mechanisms for Tool Vendors to allow modelers to expose the requirements for the objects filling their ProcessRoles in the Model, and allow code to be generated that satisfies these requirements, rather than having programmers write magic bootstrapping code.

5.2 Notification-based Mapping for the Business Process Model

In addition to the base interfaces defined in Section 5.1, the following implementations must be provided for the elements in a Business Process Model. We envisage that they will eventually be implemented as CORBA Components with a separate facet for each of the interfaces required to be supported. However, in the absence of a Component-based ORB, they will usually be implemented by a number of cooperating servants in the same address space that each expose one or more object references. The desire to avoid name mangling of element names from the model to avoid operation and attribute name clashes means that multiple inheritance is impossible in some cases.

In this mapping, a number of model elements are subsumed into behaviors of the mappings of other model elements. The general approach is that DataFlows between Ports are implemented as Structured Event transmissions between Activities. As any ProcessFlowPort usage may be a source or a sink for a DataFlow, all the mapping is done at the level of the abstract model elements ProcessMultiPort (represented by a PortUsage in the Activity which instantiates it) and ProcessFlowPort (represented by ProcessPortConnector). The conditions under which DataFlows are transmitted, and the semantics of the arrival of a DataFlow are well defined in the Business Process Model, and this mapping (as well as the Interface-based mapping in Section 5.3) concentrates only on the method of transmission of DataFlows.

5.2.1 CompoundTask (as represented by Activity)

5.2.1.1 DataFlow source

Each Activity which directly contains DataFlow sources must implement the CosNotifyComm::StructuredPushSupplier interface and connect to a Notification Channel created for the use of this BusinessProcess instance. The mapping of a DataFlow source's ProcessFlowPort (represented by a ProcessPortConnector) (Section 5.2.2) prescribes the events types to be emitted by the Activity to represent the DataFlows that these ports are sources for.

5.2.1.2 DataFlow sink

Each Activity which directly contains DataFlow sinks must implement the CosNotifyComm::StructuredPushConsumer interface and connect to a Notification Channel created for the use of this BusinessProcess instance. It must create and attach a Filter to the ProxySupplier of the Event Channel to which it is attached. The mappings of a DataFlow sink's ProcessFlowPort (represented by a ProcessPortConnector) (Section 5.2.2) provides constraints to be added to the Filter to ensure that the events representing DataFlows will be consumed at these sink elements.

5.2.2 ProcessFlowPort (represented by ProcessPortConnector)

5.2.2.1 DataFlow source

Any ProcessPortConnector, representing a ProcessFlowPort, that is the source of a DataFlow, will create and transmit a Structured Event of the following type using the Event Channel to which its containing Activity is connected.

```
domain = "EDOC"
name = "data_flow"
properties =
     flow_id : string // contains the data flow's fully qualified name
    source : string // contains <FlowPort's fully qualified Name>
    parent : string // contains <Containing Activity's Instance ID>
    payload : any  // contains the value(s) of the DataElement
```

Note that the flow_id for an ordinary flow is fixed in the model, and in the event it is scoped by the source property.

5.2.2.2 DataFlow sink

Any ProcessPortConnector, representing a ProcessFlowPort, that is the sink of a DataFlow, must create a subscription to add to its containing Activity's Filter which subscribes to the event type EDOC/data_flow, and has a constraint which selects events with the right flow_id, and source name. The parent property must also be the same as the parent of the Activity containing this ProcessPortConnector.

5.2.3 Activity(representing a CompoundTask with a Composition)

5.2.3.1 ExceptionGroup handling

An Activity representing a CompoundTask with a Composition has responsibilities in addition to those of leaf node Activities. Each such Activity must have a subscription to Events of the EDOC/exception type. The only constraint is that the exception event was emitted by a Activity instance contained directly by this Activity. This can be expressed as:

```
"parent == <My Instance Id>"
```

Upon receipt of such an event the Activity must terminate all its contained Activities, and then pass the payload of the event to the ProcessPortConnector in its PortUsage representing its CompoundTask's system ExceptionGroup.

5.2.4 ExceptionGroup

PortUsages representing ExceptionGroups are special OutputGroups that indicate a failure in the Activity that contains them. An Activity's ExceptionGroup may either be *handled*, or the data values from its Outputs must be propagated to its containing Activity's system ExceptionGroup.

If an ExceptionGroup is unhandled, that is its ProcessFlowPorts are not sources for any DataFlows, then the following event type, which will be subscribed to by the containing Activity, must be emitted when the PortUsage representing the ExceptionGroup is enabled:

```
domain = "EDOC"
name = "exception"
properties =
    source : string // contains <ExceptionGroup's fully qualifiedName>
    parent : string  // contains <Parent's Instance ID>
    payload : CosNotification::PropertySeq  // contains name/value
                                            // corresponding to its Outputs
```

If an ExceptionGroup is handled (any of its Outputs are the source of a DataFlow), then it only emits ordinary EDOC/data_flow events as specified in Section 5.2.2.1.

5.3 Interface-based Mapping for the Business Process Model

This section is an alternative to the mappings provided in Section 5.2, but it still requires the mappings in Section 5.1 as a basis.

The approach taken in this mapping is to implement all DataFlows as invocations on operations representing DataFlow sinks. The source of the DataFlow therefore is represented as an object reference to the PortUsage representing the ProcessMultiPort containing these sink points. To facilitate design (and mapped implementation) re-use, every potential DataFlow sink (i.e. every ProcessMultiPort) will be represented as an interface, so that the only runtime configuration required is the finding of object references to the objects implementing PortUsages instantiating these ProcessMultiPort interfaces. The ProcessFlowPort usages in the ProcessMultiPort instances then contain operations represent the sinks to the actual DataFlows in the Business Process Model.

5.3.1 Activity (representing CompoundTask instance)

A CompoundTask's external port contract is represented by an IDL interface type, and an Activity is implemented as an instance of this type, including the inherited Workflow interfaces defined in Section 5.1, and an object reference for a PortUsage representing the ProcessMultiPorts contained by the Activity's defining CompoundTask.

5.3.1.1 Containment

An Activity may need to be aware of its parent:

```
module EDOC {
        interface TaskNavigation {
            TaskNavigation my_container();
        };
};
```

The role of an Activity in this mapping is to provide a NameSpace for the objects it contains. As before, CompoundTasks are the IDL types we define, and Activities will the object instances at runtime. In a CORBA interface mapping this is done via modules:

```
module <CompoundTask Name> {

    interface <CompoundTask Name>Navigation : EDOC::TaskNavigation;

    // interface definitions for contained DataGroups go here

    // statically generated DataFlow sources interface goes here

    // interface containing attributes pointing to contained
    // Activities go here

};
```

5.3.1.2 DataFlow source

All CompoundTasks support a generic interface that allows their runtime Containers to provide them with the object references that they require to send out their DataFlows.

```
module EDOC {
    interface TaskDataFlowSource {
        add_data_flow_sink(
                        in string source_data_element_name,
                        in Object sink_obj_ref,
                        in string sink_data_element_name);
    };
};
```

This allows a DataFlow to be described in terms of the source name (of the form ProcessMultiPortName::ProcessFlowPortName), and the object which defines its sink ProcessFlowPort, as well as the name of the method to be called on that object. The method to be invoked is named the same as the sink ProcessFlowPort, and it always has a single parameter called "values", which is of the same type as the source DataElement.

In addition, the mapping may generate static interfaces of the form:

```
interface <CompoundTaskName>Sources {
    add_<ProcessFlowPortName>_sink(
        in <CompoundTask Module>::<sink ProcessFlowPort Name> sink);
    // etc...

};
```

There will be an operation per DataFlow for which this CompoundTask is a source. The generated code will be able to statically invoke the right operation on the sink object reference passed in to each of these operations.

5.3.2 ProcessMultiPort

5.3.2.1 ProcessFlowPort Container

A ProcessMultiPort contains a fixed (possibly empty) set of ProcessFlowPorts, each with a unique name. The following interface is generated to map the ProcessMultiPort:

```
interface <ProcessMultiPort Name> {

    // Contained ProcessFlowPort Mappings go here

};
```

There is no distinction in the interfaces between synchronous and asynchronous DataGroups; the objects implementing the interfaces must provide the appropriate semantics.

5.3.3 ProcessFlowPort

Each ProcessFlowPort is represented as an operation of the form:

```
void <ProcessFlowPort Name> ( in values <type attribute mapping>);
```

The type of the "values" parameter should be a collection type (i.e., a sequence) to support ProcessFlowPort multiplicities other than {1,1}.

5.3.4 CompoundTask (instantiated to give Activities)

5.3.4.1 DataFlow Container

A CompoundTask contains all the DataFlows that connect the Activities which it contains. This means that the Activity instantiating the CompoundTask interface is responsible for passing object references of the ProcessMultiPort interface instances (which are sinks of DataFlows) to the Activity containing the PortUsages representing the ProcessMultiPorts that are the sources of these DataFlows.

It may do this by making calls to the generic TaskSource interface (assuming that Activities making calls can use the DII), or to the statically typed generated

<*CompoundTaskName*>Sources interfaces that may be generated after this CompoundTask's usage context in the Model is known.

5.3.4.2 Exception Catcher

All CompoundTasks support an interface derived from:

```
module EDOC {

    interface CompoundTask  {
        void system_exception(
                        in payload CosNotification::PropertySeq);
    };
};
```

The interface is defined as:

```
interface <CompoundTask Name>Compound :
                                EDOC::CompoundTask;
```

The payload contains a Property for each ProcessFlowPort in the ExceptionGroup.

5.3.5 ExceptionGroup

Unhandled ExceptionGroups must call the system_exception() operation on their Container's CompoundTask interface. Handled ExceptionGroups (ones with DataFlows proceeding from their Outputs) behave the same as ordinary OutputGroups.

5.3.6 BusinessProcess

5.3.6.1 Containment

In this mapping a Business Process indicates which Task Containment level (indicated by its realizes association with a CompoundTask) is significant enough to give an outer-level module scope to.

```
module <BusinessProcess Name> {

        // realized CompoundTask declarations go here
};
```

6. Mapping from EDOC Business Process to FCM

6.1 Overview of FCM Concepts

The Flow Composition Model (FCM) is presented in part I, chapter 5 section 2.

6.2 Mapping from the Business Process Profile to the FCM

This section describes mappings from the Business Process Profile to the FCM Profile. This is shown by means of

- A mapping for each of the concrete EDOC elements
- a diagram demonstrating the use of FCM concepts to draw the procurement example introduced in the business process profile section.

6.2.1 Mapping CompoundTask

A CompoundTask is mapped to an FCMComposition, allowing it to compose the FCMNodes and FCMConnections that result from the mapping of the CompoundTask's Activities, ProcessPortConnectors, DataFlows, and ProcessRoles.

If an Activity 'uses' a CompoundTask, then it maps to an FCMFunction that is 'performed_by' a dummy FCMComponent whose sole purpose is to bind to the FCMComposition mapped by the CompoundTask. Alternateively, if the Activity is 'performed by' a PerfomerRole, it is mapped to an FCMCommand that is 'performed by' the FCMcomponent mapped to by the PerformerRole.

6.2.2 Mapping Activity

An Activity is mapped to an FCMFunction, allowing it to have as its interfaces the FCMTerminals that result from the mapping of the Activity's ProcessPortConnectors, and to be performed by the FCMComponents that result from the mapping of its ProcessRoles and the CompoundTask that it uses.

6.2.3 Mapping ProcessPortConnector

A ProcessPortConnector is mapped to an FCMTerminal, unless the ProcessFlowPort to which it refers is a ProcessMultiPort. If this is the case, the ProcessPortConnector is not mapped, although it does have implications as detailed in the mapping for the various concrete subtypes of ProcessMultiPort.

If a ProcessPortConnector is attached to an Activity, then the FCMTerminal that it maps to will be attached to an FCMFunction. If a ProcessPortConnector is attached to a CompoundTask, then the FCMTerminal that it maps to will be attached to either an FCMSource or an FCMSink, depending on whether the direction attribute of the ProcessPortConnector's represented ProcessFlowPort is 'responds' or 'initiates' respectively.

6.2.4 Mapping ProcessFlowPort

The mapping of a ProcessFlowPort depends on its direction attribute. If the direction is 'responds', then the ProcessFlowPort maps to an FCMSource, if the direction is 'initiates', the ProcessFlowPort maps to an FCMSink.

6.2.5 Mapping DataFlow

A DataFlow maps to an FCMControlLink and an FCMDataLink. The source and target of the FCMDataLink are the FCMTerminals that result from the mapping of the DataFlow's source and target ProcessPortConnectors. The source of the ControlLink is the FCMTerminal that results from the mapping of the DataFlow's source ProcessPortConnector. The target of the ControlLink is the FCMNode that holds the FCMTerminal that results from the mapping of the DataFlow's target ProcessPortConnector, unless indicated otherwise by the mapping for InputGroup, OutputGroup, and ExceptionGroup.

If more than one DataFlows emerge from a single PPC, then this is mapped to a single FCMControlLink, which leads into an FCMBranchNode, from which will emanate the newly-mapped FCMControlLinks.

6.2.6 Mapping InputGroup

The AND semantics over the ProcessPortConnectors of the InputGroup is expressed by inserting an (FCMJoinNode,FCMControlLink) pair on a path between the incoming FCMControlLinks and the FCMNode which holds the target FCMTerminals.

Asynchronous InputGroups can be mapped on a special kind of FCMTerminal – that reflects asynchronous behaviour semantics. However, since there is no explicit inclusion of such an FCMTerminal, we leave this unspecified. In the example attached, these are shown with separate rounded symbols, in a similar manner as in the EDOC Business Process notation.

6.2.7 Mapping OutputGroup

The AND semantics over the ProcessPortConnectors of the OutputGroup is expressed by inserting an (FCMControlLink, FCMBranchNode) pair on a path between the FCMControlLinks emanating from the FCMTerminals mapped from the OutputGroup's ProcessPortConnectors, and the target FCMNode.

Asynchronous OutputGroups can be mapped on a special kind of FCMTerminal – that reflects asynchronous behaviour semantics. However, since there is no explicit inclusion of such an FCMTerminal, we leave this unspecified. In the example attached, these are shown with separate rounded symbols, in a similar manner as in the EDOC Business Process notation.

6.2.8 Mapping BusinessProcess

A BusinessProcess is mapped as an FCMComposition.

6.2.9 Mapping ProcessRole

A ProcessRole is mapped as an FCMCommand and a corresponding FCMComponent.

6.2.10 Mapping Performer

A Performer is mapped as an FCMCommand and a corresponding FCMComponent.

6.2.11 Mapping Artifact

An Artifact is mapped as an FCMCommand and a corresponding FCMComponent.

6.2.12 Mapping ResponsibleParty

A ResponsibleParty is mapped as an FCMCommand and a corresponding FCMComponent.

6.2.13 Procurement Example

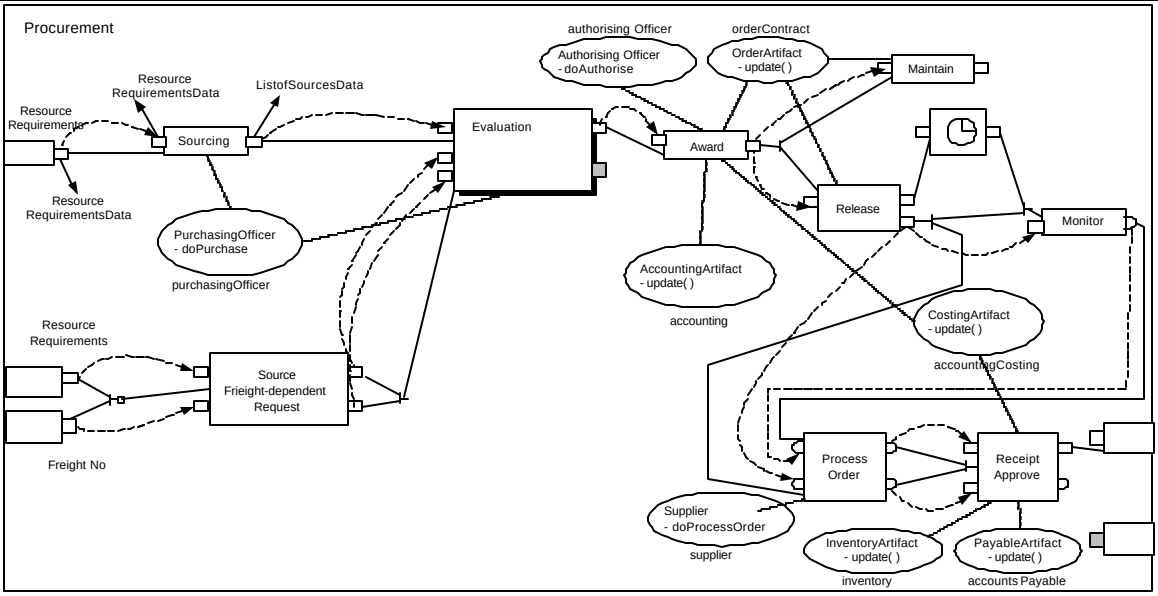


Figure 19 Procurement example