

# A Method for Developing Component-Based Applications: Role Based Coordination Component Model (RBCCM) Approach

Ninat Wanapan and Somnuk Keretho  
Department of Computer Engineering, Kasetsart University  
Email: g4365014@mcpe.ku.ac.th and sk@ku.ac.th

## Abstract

In this paper, we propose an approach for the development of component based applications based on Role Based Coordination Component Model (RBCCM) that is embedded into applications during software design. The method is general enough to support the conceptual level of the component-based software development. It provides guidelines that ease developers during the analysis and design phases. These guidelines are defined as processes for developing increasingly detailed models of constructing software systems. The RBCCM is driven by the following models: the use-context model, the role model, and relationship model. The role model abstracts system behaviors as a computational organization comprising various role relationships. The use-context model focused on exposing the design of software components to be separate from their execution contexts. These separate concerns – *computation, coordination, and policies* imposed on a given use-context—assemble the principal concept of our approach. A case study using the proposed method has been demonstrated in order to provide the feasibility of the using our approach.

**Keywords:** Methodology; Approach; Component-Based Software Development; Coordination; Role Model

## 1. Introduction

Building scalable and flexible enterprise systems using component-oriented programming has to focus on exposing the design for customization and composition of component-based software to be separated from the design of implementation of components. The former concentrates on coordination processes and additional policies imposed on component communications between participants and the latter brings together computation that represents core functionalities provided by a given component. In this paper, role based component coordination model (RBCCM) has been proposed as a model that embeds coordination model into the component software architecture. It abstracts system behaviors as a computational organization comprising various role relationships. The design of software components is explicitly defined to be separated from their execution contexts. These separate concerns, including computation and coordination as well as the policies imposed on participating software component's use-context. The outcomes of our approach are abstract role based CCM, and abstract specifications to which traditional object-oriented methodologies can be applied.

The lack of commonly accepted component-oriented methodologies that support the existing middleware technologies using the standard modeling language like UML is actually the main problem for developing component based software. We still need to find a proper methodology for component-based development because all existing methodologies work only within a component [22]. Complex interactions of components are not adequately handling. Our approaches are needed to allow the set of process to support component analysis, component design, and component construction. The propose of a component-oriented approach is to assist developers in all phases of the development of a component-based software. It solves the problem of “How we define proper components for the software systems?” toward software reuse and ease configuration in middleware infrastructure.

To avoid building an approach from scratch, it has followed and extended the popular approach to include the relevant aspect of the software components such as Unified Modeling Language (UML), and traditional OO methodologies. In other words, our approach bridges the gap between the requirements statement and the conventional object-oriented methodology.

The rest of this paper is organized as follows: Section 2 describes an overview of the role based CCM and extensions of the original CCM. Section 3 gives a brief description of the Inventory systems requirements statement that is used as a case study through out the paper. Section 4 expresses the RBCCM software architecture. Section 5 expresses the framework of our approach. Section 6 present processes for analyzing RBCCM based application in the analysis phase. Next, the processes for the design phase are defined as guidelines in section 7. Section 8 discusses and compares this approach with the conventional object oriented approach. Finally, conclusion and future work are outlined in section9.

## 2. Overview of the Role Based Coordination Component Model (RBCCM) and Extensions of the original CCM

In this section, we briefly describe the original Coordination Component Model (CCM), as described in [18] and extensions of the original CCM toward Role Based Coordination Component Model (RBCCM) for supporting the design of component-based systems using middleware infrastructure. Original CCM (referred as CCM) is the descriptive of a set of components collaborating and interacting with each other using coordination entities as connectors. It focuses on coordination views of components in a given use-context. This model promotes customization and composition with abstraction mechanisms at different levels of coordination in order to fulfill software requirements. The CCM is illustrated using UML modeling language notations as illustrated in Figure 1.

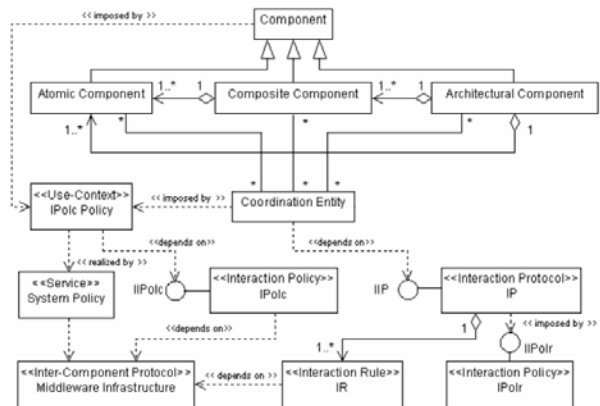


Figure 1 Original Coordination Component

From the original CMM, many problems arise when apply the model to the real world software systems such as the following items: “How can we define the CE from the software systems?”,

“What is the starting point of the software development process?” and more. We met these problems when attempted to apply the model to problems.

## 2.1 Role Based Coordination Component Model

Due to many problems as discussed previously, we extend the original CCM for problems solving. Our extended model can remedy them and be described as follow.

We extend the original Coordination Entity (CE) by realize them from many role interfaces (IRole) that inherits all related role type interfaces (Role). We will describe them in more details in the next two topics for the Component Structure and the Role Based Coordination Component Structure.

We call the extended CCM as Role Based Coordination Component Model (RBCCM) and illustrated this model by using UML modeling language notations as illustrated in Figure 2.

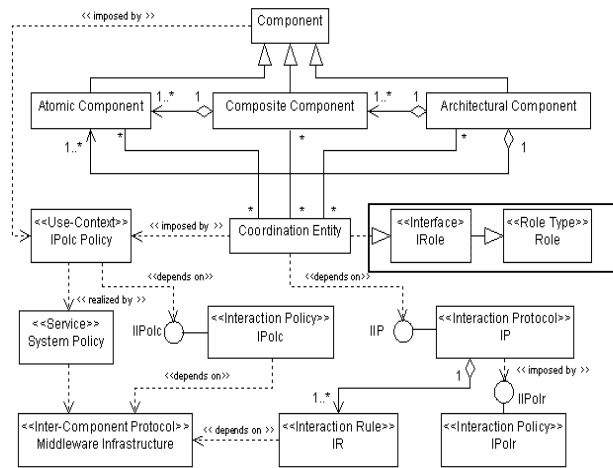


Figure 2 Role Based Coordination Component Model

## 2.2 Component Structure

A Component is a unit of deployment and a unit of composition with contractually specified interfaces, and explicit context dependencies is a must [22]. Component can be structured as atomic, composite, and architectural components [22].

- *Atomic component* can simply contain only one or more classes. It is a first level that describes a black box encapsulation, which contains a set of related classes that are packaged together with a set of immutable resources. Atomic components are the elementary units of deployment, versioning, and replacement.
- *Composite Component* contains other components as its sub-components, which can themselves be atomic or composite components. It is a second level comprising a set of relevant components participating in well-understood design patterns.
- *Architectural component* higher level encapsulations that allow reusability and scalability of the design and concepts.

## 2.3 Role Based Coordination Component Structure

The coordination component structure in Figure 2 can be described as follows:

- *Coordination Entity (CE)* The CE is an architectural component that represents domain type. This component

is a named implementation of one or more component interfaces. It realizes the coordination entity interface (IRole) that inherits all associated role type interfaces (Role). A coordination entity is a first entity that consists of abstract activities represented by component interfaces. It is used to model the domain problem and to build the system using component structure. It consists of the interaction policy that abstracts the provided services in a given context and role customization, while the protocol is the policy implementation that abstracts the interaction behavior among relevant components. It encapsulates a set of rules that governs how components behave and communicate with each other. The coordination entity governs participating components (via interaction protocols), and it may comprise components that are configured once at the coordination layer. The use-context also activates components in the model at the middleware infrastructure level.

- *Interaction Policy (IPol)* IPols are respectively managed into two aspects: first, authorizations or permissions are separated into a *use-context policy* (IPol<sub>c</sub>); second, any additional constraints imposed on the component communications are explicitly expressed as a *role interaction policy* (IPol<sub>r</sub>). IPol<sub>c</sub> is a policy imposed on the use-context and then realized by *system policy services* (e.g., ORB service such as transaction service, security service) at the ORB service level. Thus, it provides the context customization by customize the ORB such as COM+, and J2EE. IPol<sub>r</sub> defines a policy service that is imposed on the interaction protocol and expresses interaction patterns and constraints enforced on the interaction among constituent components. Thus, it supports customization of the role interaction such as pre-conditions and post-conditions of interactions.
- *Interaction Protocol (IP)* IP is an abstract unit of interactions representing the *coordination process* that manages relationships and constraints at the protocol level. It is the implementation of organization behaviors that reflects business activities and is specified separately from the interaction policy in a given use-context. Thus, it defines the behaviors that are imposed on the components interaction.
- *Interaction Rule (IR)* IR is an abstraction unit of actions that must be accomplished in order to comply with a given interaction protocol. It reflects a business process. It is explicitly specified as an encapsulated entity that transforms coordination patterns into manageable actions. Normally, coordination processes are more complexity and hard to be handled. Therefore, an interaction protocol may compose of a set of participating interaction rules, which are associated with one another to expose a given activity.
- *Inter-Component Protocol* It is the realization of interaction rules that provides application architecture and programming techniques based on middleware infrastructure. It enables an explicit separation of protocols and rules from application codes. The implementation techniques used to support rule and protocol at this level should be *scripting language*. Thus the coordination process, which is encapsulated by the interaction rule, is explicitly extended to the objects collaboration layer provided by the traditional object-oriented approach. It also uses ORB bus that provided by standard middleware such as CORBA and COM+.
- *Middleware Infrastructure* The middleware infrastructure represents components by solutions based on Object Request Broker (ORBs) such as CORBA and COM+. It

provides a mechanism for allowing interactions between applications or components in heterogeneous environment (different places or networks). Middleware mediates among software components by transforming data and coordination actions. It also supports network transparency, built-in security and transactions services, message queuing, and bridging with legacy systems.

### 3. Background of the Case Study

We consider the inventory control system and purchase system as a case study through out this paper to demonstrate how to develop component-oriented application by applying the propose RBCCM based software architecture. These two systems are used as an example of a computerized accounting system. The propose is to design the software to support both human inventory officer and accounting officer. An inventory officer is the holder of one or more account in a system can make various types of transactions upon his/her own account(s) such as add new item, adjust item balance, and update a set of attributes of item. A purchase officer can purchase new or existing into the company.

For reasons of brevity, we focus on a general perspective of the analysis and design processes and omit some details. In addition, we consider only an inventory officer who is authorized to manipulate an inventory system. An account manager can also perform the officer duties, only the manager has an additional permission to approve the following duties: a purchase request, an inventory adjustment. The inventory officer use case diagram is illustrated in Figure 3 with two actors, inventory officer and accounting manager and two generalized use cases for the InventoryOfficerDuties and ManagerOfficerDuties which comprise and additional use cases capturing business activity for each actor.

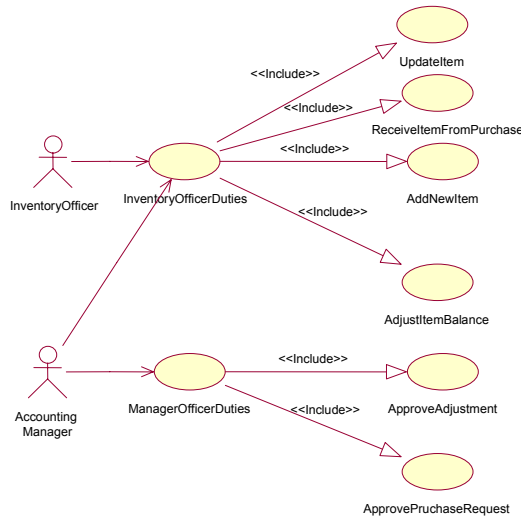


Figure 3 The inventory system use case model.

### 4. RBCCM-Based Software Architecture

In our role based coordination component model (RBCCM), the software architecture is based on the principle of separation of concerns concept comprising *computation*, *coordination*, and *context policy*. Computation provides core functionalities via specified software component. Coordination provides component framework for managing dependencies among activities represented by coordination process embedded in a role based CCM based software component such as coordination entity. Context Policies are constraints imposed on related use-context and component

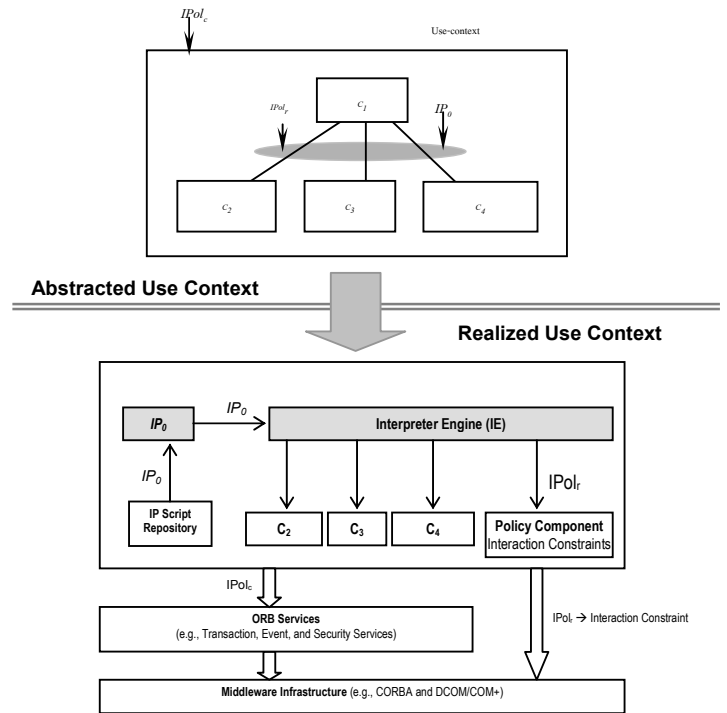


Figure 4 Use-context realizations by IE

communication. A coordination process may be implemented using either traditional programming languages or scripting language technologies. In RBCCM based software architecture, the Interpreter Engine (IE) represents an instance of any script engine that provides twofold services. First, it drives a coordination process (defined as an interaction protocol) by interpreting a related script and interacts with participating software components in order to establish required activities. Second, it implements inter component communication constraints (such as pre-conditions or post-conditions). The use-context policies are managed by the ORB services (such as transactions, security) and the role interaction policies are controlled by the IE. The interaction policies are managed into two parts: first, authorizations and permissions are separated into a use-context policy ( $IPol_c$ ) managed by the underlying middleware services; second, any additional constraints imposed on the components communications are explicitly expressed as a role interaction policy ( $IPol_r$ ).

RBCCM utilizes standard middleware infrastructures that provide system-level services and lead to the following benefits: improved systems scalability, enhanced flexibility of context customization (e.g., security service supports authentication, access control lists), and simplified programming (aware to business domain only). The use-context policy ( $IPol_c$ ) provides context customization that explicitly separated the coordination process and component communication to be managed by the system policy services.  $IPol_c$  and  $IPol_r$  realization are illustrated in Figure 4.

An interaction protocol can be implemented either by any visual programming language or by embedding a protocol component containing a *script* that governs sequence and pattern of interactions among participating RBCCM-based software components. In addition, a role interaction policy embedded as a policy component that constrains the component interactions behavior can be realized as show in Figure 5. An interaction protocol in the RBCCM architecture may be implemented as a script stored in the repository, e.g., database, directory service. By controlling a role interaction protocol as an implementation of additional policy components, the

flexibility for customization applications is provided as a routine of component software replacement.

Interpreter Engine (IE) is a generic runtime mediator that manages coordination processes and controls policies imposed on the component communications.

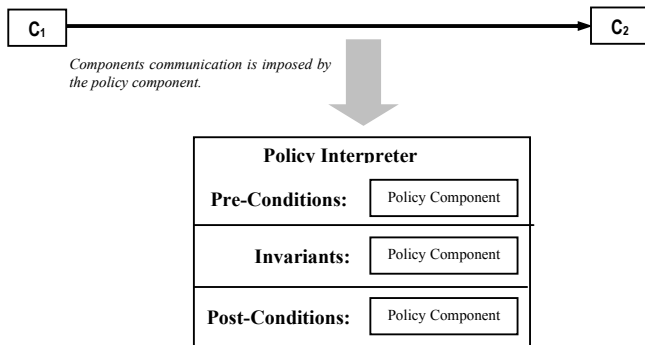


Figure 5 Interpreter Engine (IE) controls the role interaction policy according to the constrained policy components.

The IE is a general concept and can be realized by any scripting technologies such as Microsoft Windows Scripting Component [17].

### 5. Conceptual Framework of the proposed approach

In our approach, the analysis and design is a continuous process of developing detailed models of the system to be constructed. During the component-based software development, each step refines and introduced more implementation details related to the requirements captured in the use case model as illustrated in Figure 6. The methodology starts from the conceptualization phase like any traditional object oriented methodology by performing survey, feasibility study, and requirement capturing. It is then build the use case model, which captures all potential requirements in the problem domain. The method put the organization structure into the analysis and design of developing systems by including the *role*

*model and role relationship model.*

The method is then driven by the use-context model approach that captures systems features, behavior, and its environment during the analysis phase and transforms these abstract models into the concrete role based coordination component model (RBCCM) during the design phase. RBCCM focuses on exposing the design of software components to be separated from their execution contexts. These separate concerns, *computation, coordination, and policies* form the principal concept of our approach. Finally, the RBCCM is provided as a reference model that embeds the coordination model associated with the distributed computing environment within the design process. Thus, it utilizes the existing middleware infrastructure including CORBA and COM+. Hence, the benefits of developing enterprise software systems using role bases CCM consist of the software architecture to support separate concerns concept, the flexible to handle changing requirements, the design of software component to be a unit of composition with well defined and published interfaces, and the context dependencies to be explicitly defined.

The outcomes of our approach during analysis phase are the role model, role relationship model, and use-context models. These models are then refined to be models and specifications that are more concrete during the design phase. The RBCCM specifications include *component, coordination process, and policy* specifications that reflect the principal of separate concerns defined in the use-context model. And then can be implemented by the conventional object-oriented methodology. In addition, we chose the standard notation of the Unified Modeling Language (UML) and extended it to describe the RBCCM model elements. This implies seamless integration of the existing object-oriented system.

### 6. Analysis Phase

Our approach enhances the traditional OO analysis stage by providing three more models that directly reflect the organization structure of the problem domain including the *role model, role relationship model, and use-context model*. We start the analysis by using the role model to identify the key role in the system. Then, we use the role relationship model to define the role characteristic from the role model by determining responsibilities, permissions, and additional constraints imposed on both the role responsibilities and the inter-role communication. The use-context model is used to define the use-context policies (IPol<sub>c</sub>) and additional constraints, imposed on the role communication, as well as the role interaction policies (IPol<sub>r</sub>). Initially, the identified role characteristics and behaviors are captured and defined in the role type specification. As we move to the next model, the more details associated with that role is refined and accumulated in that role type specification. Finally, we have all details to establish the final role type specification.

#### 6.1 Role Model

The role model specifies the key role characteristics in the organizational structure including its responsibilities, permissions, and constraints imposed by the policies as defined in the role meta-model (see Figure 7). A role is a modeling concept that is defined as an identifier for a behavior, which can be realized by a related software component. These properties are initially captured by the *role type specification* (see Figure 8) and evolve through the *role relationship* and *use-context* models. The role model also determines role type for the role model composition in case if there is an actor who plays more than one role at a time. The role model describes precisely all the roles that constitute the organization and their positions in that organization. In an organization structure context, a complex system can be viewed as an organization that is

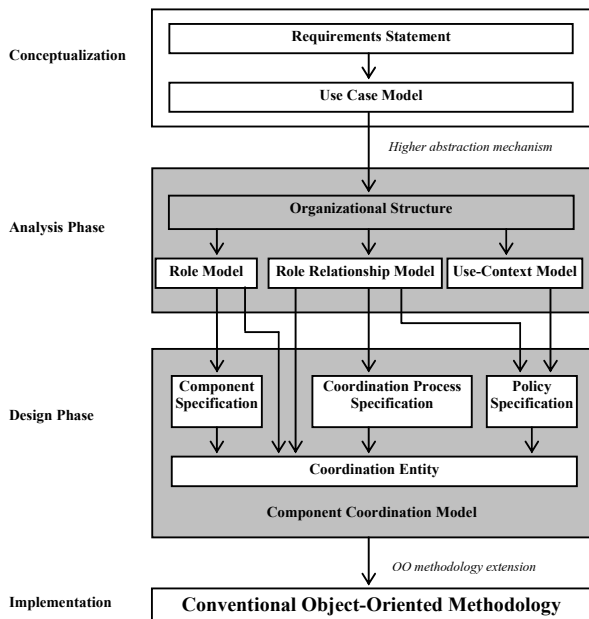


Figure 6 The methodology and the relationships between models.

defined as a collection of roles. A role is a first-class entity that provides an abstraction for capturing responsibility of a given business object in terms of its duties and rights. It is a collection of responsibilities and permissions relating to a position, which is a designated location in the structure of an organization or a social system. Thus duties and rights are associated with each role.

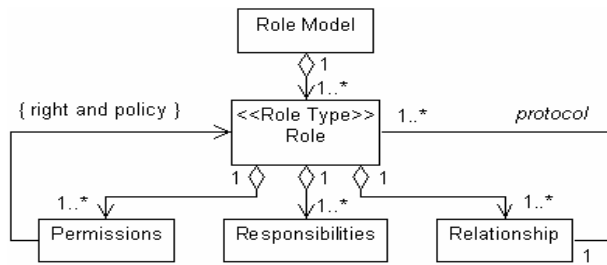


Figure 7 The Role meta-model.

<b>Role Type:</b>	a name of a given role in the role model
<b>Description:</b>	a short description of the role and its characteristics.
<b>Responsibilities:</b>	the certain functionality and the obligations that a role has to perform.
<b>Constraints:</b>	<ul style="list-style-type: none"> <li>- pre- and post-conditions.</li> <li>- invariants.</li> </ul>
<b>Relationships:</b>	<ul style="list-style-type: none"> <li>- <u>Use case relationship.</u> <ul style="list-style-type: none"> <li>- The relationship between use cases and the relevant <i>role</i> responsibilities is defined in the role type specification.</li> <li>- The relationship between an actor and the role that the actor plays is defined in the role type specification.</li> </ul> </li> <li>- <u>Role communication relationship.</u> <ul style="list-style-type: none"> <li>- The relationship between participating role types collaborating with one another in order to fulfill provided role responsibilities—the <i>inter-role communications</i>.</li> <li>- The relationship between participating role types and relevant role model compositions—the context dependencies.</li> </ul> </li> </ul>
<b>Permissions:</b>	define explicit policies associated with a given role and its context. <ul style="list-style-type: none"> <li>- a use-context policy imposed on the role responsibilities.</li> <li>- a role interaction policy imposed on the inter-role communications.</li> </ul>

Figure 8 The Role type specification templates.

In our work, a role is a higher abstraction concept for helping us identify internal and external aspects of the large and complex enterprise software systems in terms of the organizational roles and their responsibilities, role relationship, policies associated to each particular role. A software component realizing a role is required to maintain certain substantial constraints while executing. These constraints are called *role-constraint* and can be specified in the term of pre-conditions, post-conditions, and invariants. The task steps to process a new role model is:

*Task 1: Identify the key roles type in the system.*

The key role characteristics of organizational entities and their properties are captured in the related role type specification. Role responsibilities are then defined with respect to an organizational position. An organizational structure gives the perspectives that make the analysis and design of the complex system to be easier and more manageable than traditional approach. To identify roles in a system, the following aspects will typically correspond to the roles represented in the organizational structure. (a) Organizational positions representing individuals acting either within an organization. (b) Organizational structure such as divisions, departments, and business units. (c) Organizations (as a whole).

*Task 2 Construct the role model*

A role model precisely describes all the roles participating in a given use-context as well as the dependencies between them. We must define role types for the role model composition. An instance of a given runtime software component that realizes a particular role type or a collection of role types is presented by a coordination entity (CE) that plays this particular role those type is that specified role types. Role composition is realized by composing all role types that the coordination entity (the actor) has to play.

*Example:* We consider the inventory systems use case model captured during the conceptualization phase and derive it into the role model as shown in Figure 9

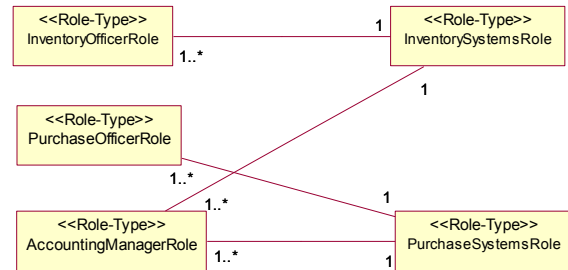


Figure 9 Inventory system role model.

**6.2 Role Relationship Model**

Role relationship model are interactions and interdependencies associated with participating roles in order to fulfill goals. Each well-defined purpose of role relationships is a role collaboration task managed by a related coordination process. Therefore, it represents a business process with regard to its associated roles. The composition of all role collaborating tasks becomes the overall business processes that manage particular resources in order to fulfill the organizational goals. The resources are role instances or business object that are used or produced within the business or organization. A role relationship model specifies the relationship between the actors of the use cases and the role types. An actor playing a particular role determines his/her responsibilities to be bound to the relationships between that role and its relevant participant. Thus, it defines patterns of interaction that must be carried out by that role in order to ensure the responsibilities. These duties have to be executed according to the permissions associated with the specified actor who play that particular role in a given use-context. The intent of the role relationship model is twofold: first, to capture the relationship between use-case model and role model; second, to capture the relationship between roles.

The sequence of activities to process a role relationship model is:

*Task 1: Determine the relationship between an actor and a particular role type that to be played by the actor (role assignment)*

The actor from the use case model and the role type that the actor plays form the role type specification are examined. Find the relationship between them in terms of role responsibilities and permissions.

*Task 2: Define the role responsibilities associated with the related use cases.*

The intention of this step is to make sure that the role responsibilities comply with the requirements captured in the use case model. Furthermore, this step assists developers to organize role responsibilities in an appropriate role type with regard to the specified use cases.

**Task 3: Define the permissions imposed on the responsibilities of a given role that a specified actor plays.**

The permissions imposed on this attribute are defined. What we try to do here is to separate the permissions as policy statements defined in the environment that all instances of the RBCCM-based components are deployed at the run-time. These policy statements imposed on a given role responsibilities are explicitly defined in the role type specification. They will be furthermore refined as use-context policies in the use-context model.

**Task 4: Determine the relationship between relevant role types in the role model.**

Determine whether there are any other role types in the role model that have dependencies with the target role type. Also, examine the related role types that form the role model composition in order to determine the context dependencies.

**Task 5: Define the role communication**

The established pattern of interactions is defined. The relationship defines the role communication with regard to the *sequences* and *orders* in which a given role has to interact with other participants. Consequently, this pattern of interactions carries out all responsibilities this particular has to provide as illustrated by the association between role types and related use cases. In other words, the role relationship model determines how business activities, captured by the use case model, associate with the related role responsibilities.

**Task 6: Define additional constraints imposed on the role communications.**

Any additional constraint imposed on the relationship between collaborating roles, such as pre-conditions or post-conditions enforce on the customer transaction, is also separated as *role interaction policy* expressed in the use-context model at the analysis phase and will be realized by the underlying policy model at the design phase.

**Example:** Figure 10 illustrates the role relationship model of the inventory system scenarios.

### 6.3 Use-Context Model

The use-context model is a logical boundary capturing policies imposed on the environment, such as authorizations and permissions with respect to the role and relationship models. It also defines the scope of interested *policies*, *protocols*, and *rules* that constraint the role communications defined in the previous models. A context

dependency expresses all contractual agreement and dependencies that a component requires in order to commit to the exposed interfaces including expected functionalities, required services, and sequences and states that participating components involve. At minimum, component specifications should provide a contract stating both the services a component offers and services it requires in order to fulfill its commitments. In the use-context model, the role meta-model in Figure 7 is derived and refined to expose two types of policies: the *use-context policy* (IPol<sub>c</sub>) and the *role interaction policy* (IPol<sub>r</sub>). The use-context policies capture and define the authorizations and permissions enforced on the role responsibilities as the IPol<sub>c</sub> specification. The role interaction policies capture and define the additional constraints imposed on the role communications as IPol<sub>r</sub> specifications. Furthermore, it exposes another important attribute with regard to the interactions and dependencies (the coordination process) which is represented by a related *interaction protocol* (IP).

The benefits of the use-context model are as follows:

*-Explicitly define context dependencies.* A software component assigned to a relevant role has the explicit context dependencies defined in the role type specification. Normally, an individual role model focuses on a single context. When a role from a specified role model is fulfilled by a given run-time component, the instance of this component corresponds only to a particular context.

*-Manage role model composition.* If an active component plays more than one role at a time, these roles have to be composed with respect to the role model composition. By explicitly separating the use-context policy and role interaction policy in the use-context model, the role model composition is easily managed through the policies management defined as another separate concern.

*-provide customization and composition toward design for change.* The use-context model focuses on exposing the design of software components to be separated from their execution context. Therefore, these separate concerns are refined as follows: computation is assigned by related component; coordination is represented by a coordination entity; and policies are provided by both IPol<sub>c</sub> and IPol<sub>r</sub>. They provide the building of enterprise software as a process of defining roles as sets of policies toward the interaction policies, protocols, and rules. Therefore, changing requirements is much easier by composing, updating, and changing these policies, protocols, and rules. Furthermore, the customization and composition of the software design is emphasized at the beginning of the analysis phase via role and role relationship models. Consequently, the RBCCM-based software components in the design phase are directly derived from the specifications defined in the use-context model.

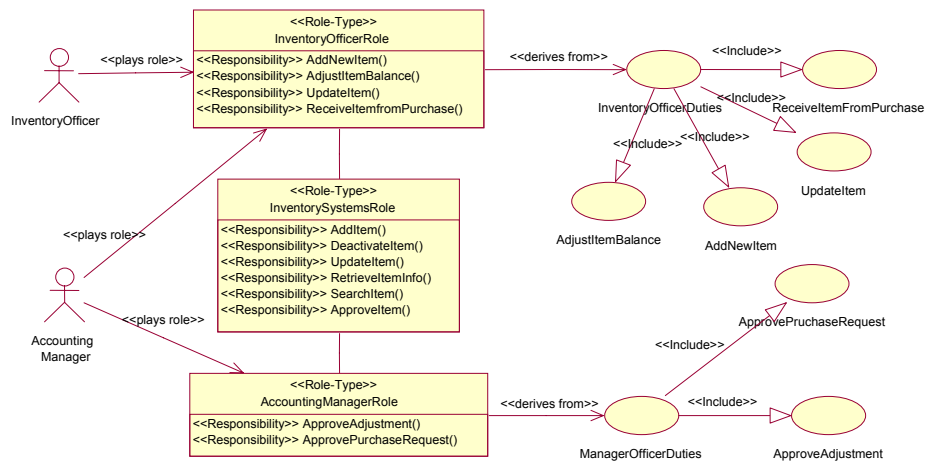


Figure 10 Role Relationship Model of the inventory system

The process steps to construct the use-context model are defined as follows:

*Task 1: Define the permissions imposed on the role responsibilities as use-context policies (IPol<sub>c</sub>).* A permission statement imposed on a given role responsibility is explicitly defined as a use-context policy. Thus, the use-case relationship and permissions are refined to be use-context policies.

*Task 2: Define the coordination processes, which manage dependencies among roles that perform activities as interaction protocol (IP).* A given role responsibility is the result of interdependencies between participating roles that form a pattern of interactions. In a complex system, a coordination process might be very complicated for understanding; therefore, an IP specification might contain additional interaction rules (IR) provided as a decomposition mechanism.

*Task 3: Define the additional constraints imposed on the role communications from the role relationship model as the role interaction policy (IPol<sub>r</sub>).* IPol<sub>r</sub> imposed on a given role communication is implemented as a RBCCM-based software component and realized by the interpreter engine (IE).

*Task 4: Refine the role-type specification by adding additional information which is discovered in the use-context models.* The use-context model helps us for the additional context information discovery. This information is used to refine the role type specification.

*Example 4: From the InventoryOfficer role type specification, we can define the use-context policies (IPol<sub>c</sub> specifications), the role interaction policy (IPol<sub>r</sub> specifications). Figure 11 illustrate the role-type specification InventoryOfficer after refined with additional informations.*

<b>Role Type:</b>	<b>InventoryOfficer</b>
<b>Description:</b>	an employee of an organization who is authorized to process inventory transaction.
<b>Responsibilities:</b>	<ul style="list-style-type: none"> <li>-Add new item into the inventory system.</li> <li>-Receive item from purchase.</li> <li>-Adjust item balance.</li> <li>-Update attributes of items.</li> </ul>
<b>Constraints:</b>	<p><b>Pre-Conditions:</b></p> <ul style="list-style-type: none"> <li>-Employee has a valid employee-system account.</li> <li>-Item has a valid item code.</li> </ul> <p><b>Post-Conditions:</b></p> <ul style="list-style-type: none"> <li>-Item balance is not below zero.</li> </ul> <p><b>Invariants:</b></p> <ul style="list-style-type: none"> <li>-Only inventory officer and account manager can play this role.</li> </ul>
<b>Relationship:</b> <b>[Interaction Protocols]</b>	<p><i>Determine the coordination processes by examine the role relationships that represents interactions and dependencies between participating roles and constraints of role-type specification attribute.</i></p> <ul style="list-style-type: none"> <li>-IP<sub>1</sub>: Add new item into the inventory system.</li> <li>-IP<sub>2</sub>: Receive existing item from purchase. <ul style="list-style-type: none"> <li>IR<sub>1</sub>: Update related purchase order paper status.</li> <li>IR<sub>2</sub>: Update the item balance.</li> </ul> </li> <li>-IP<sub>3</sub>: Adjust item balance.</li> <li>-IP<sub>4</sub>: Update attributes of items.</li> </ul>

**[Role Interaction Policy: IPol<sub>r</sub>]**

*Refine the role relationship that defines the additional constraints imposed on the role communications.*

*Not available for this case study.*

**[Use-Context Policy: IPol<sub>c</sub>]**

*Refine the use-case relationship and permission attributes of the first- cut role type specification.*

Security policy:

Only inventory officer can process inventory transactions.

Transaction processing policy:

All inventory transaction requires transaction processing service.

Figure 11 Refined role-type specifications.

**7. Design Phase**

The RBCCM design process concentrates on transforming the derived abstract models established during the analysis phase into concrete models that are detailed enough for being implemented by the underlying traditional OO approach. We deliver the role based component coordination model with three additional specifications, including *component specification*, *coordination process specification*, and *policy specification* as refinement mechanism for designing each RBCCM-based component. In Figure 6, the derived role model from the analysis phase is refined during the design phase into a component specification. The derived role relationship model is refined into a concrete coordination process specification specifying the related interaction protocol that controls the participating components. The derived role relationship and use-context models are refined into a policy specification that manages all right imposed on both the encompassed use-context and component communications. Finally, the derived role and role relationship models as well as the realization of there three specifications are constructed into a coordination entity that is, an instance of the RBCCM-based software component.

**7.1 Component Specification**

Component model transforms the abstract role model derived during the analysis phase into the constituent concrete entities according to the component structure. The associated component specification is used for role assignment. Any software component that fulfills a given component specification will guarantee to carry out the specified role type specification. Thus, the design of a component is complied with the related role type specification derived from the role model and its specification covers all required tasks or activities represented by that particular role responsibilities. A component can be atomic component that maintains the resources such as information persisting in the database or a composite component aggregating another atomic components, composite components, or coordination entities. The task steps to create a component specification are defined as follows:

*Task 1: Identify the existing component in the system.* Reuse the software component by either identifying one in the system or creating a new one. This component must conform to the RBCCM based component structure, including atomic component, composite component, and architecture component, and comply with a given role type specification.

*Task 2: Define the component model.* The model explicitly defines context dependencies including their provided and required functionalities. The former directly represents role responsibilities derived from the use case model and the latter are services required from other participants in the role model. It specifies a component that may claim to describe the behavior required by a given role type specification regardless of its implementation. Hence, the

component model directly reflects the role type specification and provides a model for defining a component specification.

**Task 4: Define the component specification** The component specification explicitly defines static behavior, component structure, and context dependencies with respect to the previous defined model and it comprises a construction that describe component structure including provided operations, constraints, and participating components. The component specification contains six attributes in its construction: (1) *Name* of component specification; (2) *Description* of the defined specification; (3) *Role assignment* defines the realization between a role type specification and a software component that fulfills that particular role; (4) *Interfaces* represent the component provided operations; (5) *Constraints* define restrictions imposed on the provided operations such as invariants, pre-conditions, post-conditions; (6) *Participants* define all required operations from the participating software components.

**Example:** From the inventory officer role-type specification (see Figure 11) in the analysis phase we can define the new *InventoryOfficer* component. The component model with respect to the *InventoryOfficer* use cases model, role-type responsibilities, and IP specifications. Each use case associated with a role responsibility that requires services represent by participating use cases (see Figure 12)

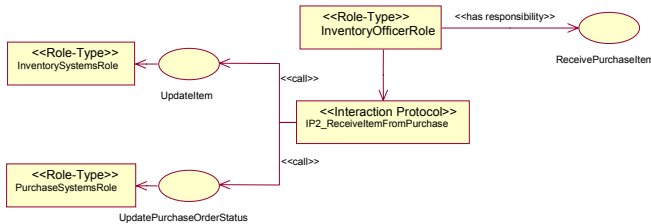


Figure 12 The relationship between use cases and a given role responsibility defines interaction protocol (IP2) as a coordination process associated with required business actions provided by participating *InventorySystemRoler* and *PurchaseSystemRole* role-type.

For simplicity, the *InventoryOfficerRole* role-type specification with regard to only *ReceivePurchaseItem* use case is illustrated. The *InventoryOfficerRole* role-type required interaction protocol (IP<sub>2</sub>) as a coordination process in order to handle the context dependencies associated with the required business actions, including *UpdateItem* and *UpdatePurchaseOrderStatus* from the *InventorySystemRole* and *PurchaseSystemRole* role-type, respectively. These required business actions may be encapsulated as scripts embedded in the interaction rules that define how the provided actions may perform. Thus, *UpdateItem* and *UpdatePurchaseOrderStatus* are established as *IR1\_UpdatePurchaseOrderStatus* and *IR2\_UpdateItemBalance*. Consequently, the interaction protocol (IP1) composes of interaction rules *IR1\_UpdatePurchaseOrderStatus* and *IR2\_UpdateItemBalance*. Next, role assignments with the appropriate software components and coordination entities must be defined. Therefore, the software component assigned to fulfill the *InventoryOfficer* role-type specification is fulfilled by the *InventoryOfficer* component as illustrated in Figure 14. The collaborating *InventorySystemRole* and *PurchaseSystemRole* role types are assigned with coordination entities *CE\_InventorySystemRole* and *CE\_PurchaseSystemRole*, respectively. They provide services that the *InventoryOfficer* software component requires. The business actions exposed via the interface *IUpdatePurchaseOrderStatus* and *IUpdateItemBalance*, respectively.

In order to have a component that can carry out this specified role, the *InventoryOfficer* component specification that fulfill the

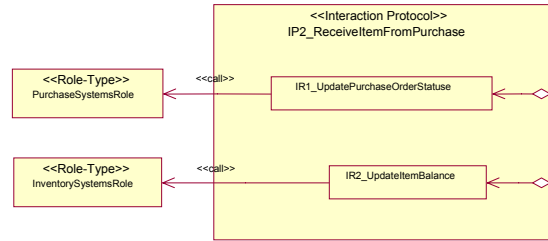


Figure 13 The interaction rules, *IR1\_UpdatePurchaseOrderStatus* and *IR2\_UpdateItemBalance*, are the embedded scripts that define behavior of the required business actions provided by the *PurchaseSystemRole* and *InventorySystemRole* role-type, respectively

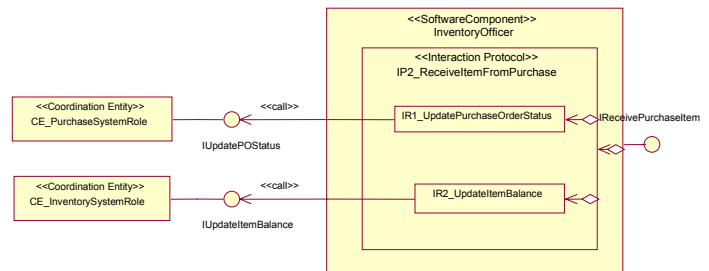


Figure 14 The role assignment, that is the *InventoryOfficer* software component fulfills the *InventoryOfficerRole* role type.

*InventoryOfficerRole* role type specification is given in Figure 15. The run-time software component that conforms to the *InventoryOfficer* component specification will be deployed in the *InventoryOfficer* use-context in order to fulfill the *InventoryOfficer* role.

Component Specification: <i>InventoryOfficer</i>	
<b>Description:</b>	an employee of a company who is authorized to process inventory transaction.
<b>Role Assignment:</b>	<i>InventoryOfficer</i> role type specification (see also Figure 11).
<b>Interfaces:</b>	<ul style="list-style-type: none"> <li>- IAddNewItem ()</li> <li>- IReceivePurchaseItem ()</li> <li>- IAdjustItemBalance ()</li> <li>- IUpdateItemAttribute ()</li> </ul>
<b>Constraints:</b>	<p>-- IP1 interaction Protocol.  <i>(for brevity reason the detail is not shown)</i></p> <p>-- IP2 interaction Protocol. Employee has a valid employee-code, item has a valid item-code, and item balance must be equal or less than maximum quantity attribute of that item.</p> <p><b>Context</b> <i>InventoryOfficerContext</i>:: <i>ReceivePurchaseItem</i> ( <i>ItemCode</i>: Item-code, <i>POCode</i> : Purchase Order-code, <i>e</i> : Employee-code) : Boolean</p> <p><b>Pre</b>: self.inventorysystem.validate (<i>ItemCode</i>:Inventory-code)</p> <p><b>Pre</b>: self.inventorysystem.authorize (<i>e</i>: Employee-code)</p> <p><b>Pre</b>: self.purchasesystem.validate (<i>POCode</i>: Purchase Order-code)</p> <p><b>Post</b>: result - self.inventorysystem.item.querybalance (<i>ItemCode</i>: Item-code) &lt;= self.inventorysystem.item.queymaxquantity (<i>ItemCode</i>:Item-code)</p> <p>--IP3 interaction Protocol.</p>

	(for brevity reason the detail is not shown) --IP4 interaction Protocol. (for brevity reason the detail is not shown)
<b>Participants:</b>	CE_InventoryOfficerRole CE_PurchaseSystem

Figure 15 The component specification of the InventoryOfficer role type specification

## 7.2 Coordination Process Specification

Coordination process specification reforms coordination processes into reusable interaction protocols (IPs) that might comprise additional interaction rules (IRs) for managing complexity of the coordination processes. It transforms the abstract role relationship model (pattern of role interactions) derived during the analysis phase into the concrete interaction protocol that normally consists of the underlying interaction rule realized by either a protocol component or a related script. These interaction protocols govern how run-time software components coordinate themselves through the coordination bus (ORB bus). They manage a set of constitute components and coordination entities in order to fulfill the required business process. We can construct the coordination process specification by following a task step:

*Task 1 Define the coordination specification.* Refine and define both IP and IR specification using UML interaction diagrams in order to exhibit the dynamic behavior of the specified coordination process. The coordination process specification comprises four attributes in its construction: (1) *Name* of the coordination process specification; (2) *Description* of the defined specification; (3) *IP specifications* that define interaction protocols that fulfill coordination processes; (4) *Interaction Diagrams* that represent the dynamic behavior of the interaction protocols.

*Example:* IP<sub>2</sub> specification defines the coordination process for receiving the purchase item activity (the responsibility *ReceivePurchaseItem* in InventoryOfficerRole role type). In order to manage the dependencies among several actions, the IP<sub>2</sub> is decomposed into two interaction rules including IR<sub>1</sub> and IR<sub>2</sub>. IR<sub>1</sub> specification manages the action, which performs updating purchase order status, and IR<sub>2</sub> specification performs updating item balance. Therefore, the InventoryOfficerRole role type has to communicate with the InventorySystemRole and PurchaseSystemRole role type for these functionalities. The IP<sub>2</sub> specification that describes the *RecievePurchaseItem* coordination process is defined in Figure 16 and the UML sequence diagram can express the coordination process as illustrated in Figure 17.

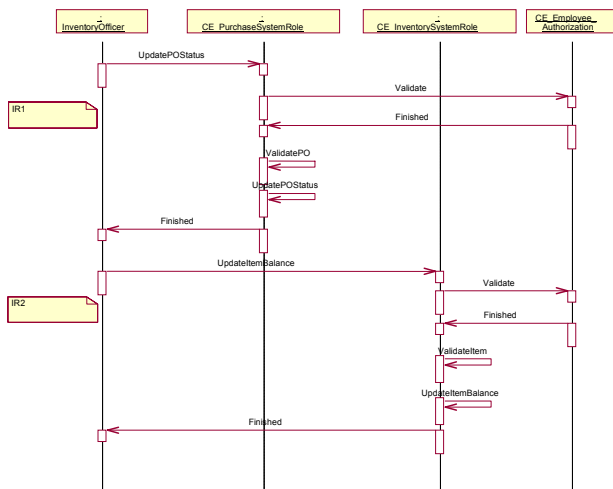


Figure 17 IP<sub>2</sub> sequence diagram represents the IP<sub>2</sub> specification, which coordinates the *ReceivePurchaseItem* process, which comprises of IR<sub>1</sub> and IR<sub>2</sub> specifications.

Coordination Process:InventoryOfficer	
<b>Description:</b>	InventoryOfficer interaction protocol comprises four business processes, which are realized by sub-interaction protocols as follows. <ul style="list-style-type: none"> <li>- Add new item transaction business process (IP<sub>1</sub>).</li> <li>- Receive purchase item business process (IP<sub>2</sub>).</li> <li>- Adjust item balance business process (IP<sub>3</sub>).</li> <li>- Update item attributes business process (IP<sub>4</sub>).</li> </ul> each business process can be realized by a collection of related interaction rules.
[Interaction Protocols]	examine the coordination process defined in the <b>Role Relationship</b> that represents interactions and dependencies between participating roles and <b>responsibility</b> and <b>constraints</b> properties of role type specification attribute. For simplicity, only IP <sub>2</sub> is described in detail.
<b>IP Specifications:</b>	<ul style="list-style-type: none"> <li>- IP<sub>1</sub> = Add new item transaction business process. (for simplicity the detail is not shown here).</li> <li>- IP<sub>2</sub> = Receive purchase item business process [Interaction Rules] update purchase order status and update item balance.  <b>IR Specifications:</b> -IR<sub>1</sub> = UpdatePOStatus. -IR<sub>2</sub> = UpdateItemBalance.  -- constraints. <b>context</b> InventoryOfficer :: UpdatePOStatus( e : Employee-Code, POCode : Purchase Order-code) : Boolean <b>pre:</b>e.validate ( e : Employee-Code ) <b>pre:</b>purchaseSystem.validatePO ( POCode: Purchase Order-Code )  <b>context</b> InventoryOfficer :: UpdateItemBalance ( e: Employee-Code, ItemCode: Item-Code ) : Boolean <b>pre:</b>e.validate ( e : Employee-Code ) <b>pre:</b>inventorySystem.validateItem (ItemCode: Item-Code )</li> <li>- IP<sub>3</sub> = Adjust item balance. (for simplicity the detail is not shown here).</li> <li>- IP<sub>4</sub> = Update item attributes business process. (for simplicity the detail is not shown here).</li> </ul>
<b>Interaction Diagrams:</b>	<ul style="list-style-type: none"> <li>- IP<sub>1</sub> sequence diagram represents <i>AddNewItem</i> coordination process</li> <li>- IP<sub>2</sub> sequence diagram represents <i>ReceivePurchaseItem</i> coordination process (not shown here). (see Figure 17).</li> <li>- IP<sub>3</sub> sequence diagram represents <i>AdjustItemBalance</i> coordination process (not shown here).</li> <li>- IP<sub>4</sub> sequence diagram represents <i>UpdateItemAttributes</i> coordination process (not shown here).</li> </ul>

Figure 16 The coordination process specification of the InventoryOfficer role type specification

### 7.3 Policy Specification

RBCCM's policy model explicitly defines a number of policies imposed on the use-context model and role communication from the role relationship model. All components are deployed in a specified use-context, which is realized by a component container that represents the run-time environment in a distributed computing infrastructure framework. This container includes a context object for each component. For instance, EJB and COM+ both provide context *ContextObjects*, which maintain specific context information such as the states of transactions, security, persistence, and deployment. Thus, the policy specification has two purposes: first, to provide information on how to configure and implement these ORB service policies; second, to expose the role interaction policy that has to be implemented at the communication. The specification can be established by following steps:

*Task 1 Review the role model and role type specification including role responsibilities.*

*Task 2 Review the role relationship model and use case relationship model including interaction protocol (IP specifications) and interaction rules (IR specifications).*

*Task 3 Review the use-context model including use-context policies (IPol<sub>c</sub> specifications) and role interaction policies (IPol<sub>r</sub> specifications).*

*Task 4 Define the policy specification.* Refine and formalize both IPol<sub>c</sub> and IPol<sub>r</sub> specifications using object constraint language (OCL).

*Example:* The InventoryOfficer policy specification expressing the IPol<sub>c</sub> and IPol<sub>r</sub> specifications is illustrated in Figure 18.

Policy Specifications: InventoryOfficer Policy	
<b>Description:</b>	<ul style="list-style-type: none"> <li>- only manager (an actor) can approve approving transaction.</li> <li>- enter customer transaction process requires transaction processing service.</li> </ul>
[Use-Context Policies]  <b>IPol<sub>c</sub> Specification:</b>	<p>refine the derived <i>Use case relationship</i> and <i>Permissions</i> attributes and precisely define the constraint using OCL (see also Figure 11).</p> <ul style="list-style-type: none"> <li>- <b>authorization policy.</b> <ul style="list-style-type: none"> <li>-- IP<sub>3</sub> interaction Protocol is executed in the ORB security service, which is</li> <li>-- implemented on the ORB container such as EJB ContextObject or COM</li> <li>-- ContextObject.</li> </ul> </li> </ul> <p><b>context</b> InventoryOfficer::ApprovePurchaseOrder (e : Employee-Code) : Boolean  <b>pre:</b> self.oclsTypeOf ( AccountingManager )</p> <ul style="list-style-type: none"> <li>- <b>transaction processing policy.</b> <ul style="list-style-type: none"> <li>-- enter customer transaction process requires transaction processing service.</li> <li>-- IPol<sub>c</sub> imposes Transaction Processing Monitor, which is realized by ORB service,</li> <li>-- e.g., EJB ContextObject and COM ContextObject.</li> </ul> </li> </ul>
[Role Interaction Policies]  <b>IPol<sub>r</sub> Specification:</b>	<p>refine the Role Relationship that defines the additional constraints imposed on the inter-role communications.</p> <p style="text-align: center;"><i>{Not available}</i></p>

Figure 18 The policy specifications of the InventoryOfficer role type specification.

### 7.4 Coordination Entity

RBCCM's coordination entity model transforms the abstract role model and relationship model during analysis phase into a component that represents a domain entity. A coordination entity realizes the interface of particular role model (or the role type specification, if any) in a specified use-context and interacts with other participating role types in order to carry out the provided activities. In other words, a coordination entity is assigned to a given role that composes of one or more role types and it is the run-time software component that fulfills that particular role responsibilities. We can define the coordination entity (CE) with regard to RBCCM reference model by following tasks steps.

*Task 1 Define coordination entities and participating software components with regard to the role based component coordination model as the reference model.*

### 8. Discussions

The proposed methodology based on the RBCCM has successfully addressed a number of issues related to the application design as follows:

*Component granularity.* RBCCM based software component provides a higher abstraction than class granularity by using component technology. Since the smallest unit of analysis and design is a component. However, the class abstraction has been used to implement software components with respect to the traditional OO approach.

*Separation of concerns.* The coordination model is embedded in the component software architecture and it supports the separation of concern including *computation*, *coordination*, and *policy* enforced on the execution environment (via use-context policy) and interaction policy (via role interaction policy). We believe that our approach explicitly facilitate contextual dependencies via separation of concerns defined in the RBCCM.

*Component complexity.* The developed RBCCM-based software components are deployed in a distributed computing environment by utilizing a standard ORB (such as CORBA and COM+). The core functionalities, coordination processes, and policy specifications could be developed concentrated to business activity only. In addition, many policies are supported by the currently available ORB services such as transaction processing service, security service, and event service. For example, the authorizations or permissions are separately configured using COM+ service utilities such as COM+ component services administrative tool. Thus, we claim that RBCCM supports the software component concept, which defines a component as a unit of independent deployment and a unit of composition with contractually specified interfaces and explicit context dependencies.

*Component coordination.* Because of separating in interaction protocol and interaction rule, a level of reusability is increased and the interoperability between participating components could be simplified toward the customization and composition approach.

*Role assignment.* The method promotes a software component that fulfills a specified role to be a unit of composition with regard to the RBCCM.

### 9. Conclusions and Future Work

The contributions of this paper are described as follow:

- It provides a formal design procedure for component based software by using role modeling and extended original CCM (we call RBCCM).
- A formalization of mappings between component roles and components.
- A syntactic and semantic specification of role, role interactions, component, coordination process, and context composition.

- The task steps and guideline that help developers build component oriented applications based on currently component and middleware technologies.

We suggest the following areas for future research:

- Realization the interpreter engine (IE) with the scripting technology for dynamic customization and composition.
- Studying the applicable of the component load balancing technique (CLB) to improve the enterprise component based software performance compensated to time that used to interpret the script of IE.

## 10. References

- [1] B.B. Kristensen. "Component Composition and Interaction." Proceedings of International Conference on Technology of Object-Oriented Languages and Systems (TOOLS PACIFIC 96), Melbourne, Australia, 1996.
- [2] B.B. Kristensen. "Object-Oriented Modeling with Roles", OOS'95, Proceedings of the 2nd International Conference on Object-Oriented Information Systems, Dublin, Ireland, 1996.
- [3] Booch, G., Rumbaugh, J., and Jacobson, I., 1999. The unified modeling Language User Guide, Addison-Wesley.
- [4] Booch, G., 1994. Object-Oriented Design with Applications, Second Edition. Benjamin/Cummings, Redwood City.
- [5] D. Riehle and T.Gross, "Role Model Based Framework Design and Integration." OOPSLA'98, Proceedings of the 1998 Conference on Object Oriented Programming Systems, Languages and Applications, ACM Press, 1998.
- [6] D'Sousa, D. F. and Wills, A. C., 1998. Objects, Components, and Frameworks with UML: The Catalysis Approach. Addison Wesley Longman Limited.
- [7] E. A. Kendal., 1999. "Role Model Designs and Implementations with Aspect Oriented Programming," OOPSLA'99 Denver, November, 1999.
- [8] E. A. Kendal., 1999. Role Modeling for Agent System Analysis, Design, and Implementation. In Proceeding of the 1st International Symposium on Agent Systems and Applications and 3rd International Symposium on Mobile Agents, October 3-6, Palm Springs, California, IEEE CS Press, 204-218.
- [9] E. Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1998.
- [10] Eriksson, H. E. and Penker, M., 1998. UML toolkit. The Wiley Computer Publishing, New York.
- [11] Eriksson, H. E. and Penker, M., 2000. Business Modeling with UML: Business Patterns at Work. John Wiley & Sons, NY.
- [12] G. Gottlob, M. Schrefl, and B. Rock. "Extending Object-Oriented Systems with Roles". ACM Transaction on Information Systems, 14(3), 268-296, 1996.
- [13] Kirtland, M., 1999. Designing Component-Based Applications. Microsoft Press, Redmond, Washington.
- [14] Liping Zhao and E. A. Kendal., 2000. "Role Modeling for Component Design," in Proceedings of the 33rd Hawaii International Conference on System Sciences.
- [15] Lupu, E. and Sloman, M., 1997a. A Policy Based Role Object Model. In Proceeding of the 1st International Enterprise Distributed Object Computing Workshop (EDOC'97), Queensland, Australia, 36-47.
- [16] Lupu, E. and Sloman, M., 1997b. Towards a Role-Based Framework for Distributed Systems Management. In Journal of Network and Systems Management, Plenum Press, 5(1): 5-30.
- [17] Microsoft, 2000. Microsoft Corporation. <URL: <http://www.microsoft.com/com/tech/complus.asp>>
- [18] Olarnsakul, M. and Batanov, D., 2000. A Component Coordination Model for Customization and Composition of Component-Based System Design. In Proceeding of the 7th Annual IEEE International Conference and Workshop on the Engineering of Computer Based System (ECBS 2000), April 3-7, Scotland, UK, 228-236.
- [19] OMG-UML v1.3, 2000. OMG Unified Modeling Language Specification, Version 1.3, First Edition: March 2000. Object Management Group. <URL=<http://www.omg.org>>
- [20] Pooly, R. and Stevens, P., 1999. Using UML Software Engineering with Objects and Components. Addison Wesley Longman Limited.
- [21] Quatrani, T., 1997. Visual Modeling with Rational Rose and UML. Addison Wesley Longman.
- [22] Szyperski, C., 1999. Component Software: Beyond Object-Oriented Programming. ACM Press, NY.
- [23] Warmer, J. and Kleppe, A., 1999. The Object Constraint Language: Precise Modeling with UML, Addison Wesley Longman.