Developing Software Applications Using Middleware Infrastructure: Role Based and Coordination Component Framework Approach

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

Abstract
This paper presents a methodology for the development of component-based software based on a role-based model and a component coordination model (CCM). This research extends the role-based model to define and abstract the business process and business relationship based on the roles that use the system. The basic idea of the CCM is to clearly separate the space of computation that is related to the internal behavior of software components from the space of coordination that is related to how components must interact with one another in a given application context. This paper also enhances the CCM model to incorporate role realizations for coordination entities. The proposed methodology provides developers with guidelines that notably simplify the design and construction of the distributed component-based applications using middleware infrastructure. We are also extending the UML notations to describe this integrated methodology.

Keywords: methodology, component-based software development, coordination, role-based model

1. Introduction
Progress in software engineering has primarily been made through the development of increasingly powerful abstraction mechanisms that model and develop complex systems. Large and complex system development confronts two main difficulties including rapid changing requirements and life cycle of software evolution. A system that can be evolved is a system that has an ability to withstand changes in its requirements, environment, and implementation technologies. Architecture capable of accommodating changes must be specifically designed for change and the overall architecture should be defined in order to provide us with how to manage and coordinate all constituent components into a desired business activity.

The development of complex distributed software systems using de facto middleware infrastructures such as OMG CORBA [3] and Microsoft DCOM/COM+ [2] involves an understanding of complicated component characteristics and their interdependency behaviors. Present software systems usually include reusable components and other third party components with different characteristics working together to fulfill required services. Software applications nowadays have to not only fulfill their functional requirements, but must also be open and flexible in a variety of ways, for instances, they might be portable to different platforms, interoperable with other applications, extendible to new functionality, configurable to individual users, and maintainable [7]. Building scalable and flexible enterprise systems using component-oriented programming have to focus on exposing the design for customization and composition of component-based systems to be separated from the design of components. The former concentrates on coordination processes and additional policies imposed on inter-component communications between participants and the latter brings together computation that represents core functionalities provided by a given component. In this research, Component Coordination Model (CCM) [3]
has been proposed as a reference model that embeds coordination model into the component software architecture. It directly reflects the role model, which abstracts system behaviors as a computational organization comprising various role relationships. It focuses on exposing the design of software components that are to be separated from their execution contexts. These separate concerns, computation and coordination, and the policies imposed on the use-context form the principal concept of our models. The outcomes of our approach were the new methodology for developing the component based enterprise systems which extended CCM framework by adding various diagrams such as the enhanced role model that can describe the role in the business and extending the UML notations to describe the business process, and finally the methodology has utilized the existing distributed technologies to realize implementation.

2. Background of the Component Coordination Model (CCM)

In this section, we briefly describe CCM, as defined in [3], for supporting the design of computer-based systems using a middleware infrastructure for incorporating related models into software systems. CCM is the description of a set of components collaborating and interacting with one another using coordination entities as connectors and it is illustrated using UML modeling language notations as illustrated in Figure 1. Application designs are collections of relevant components interacting to each other, in a given sequence, in order to fulfill requirement specifications. The design for component specification that describes core functionalities of components (exposed via Interfaces) is currently available using CORBA and DCOM Interface Definition Languages (IDLs). It realizes an implementation of interaction policies and interaction protocols, which acts in conformity with the CCM principles, as interfaces. A model is based on the customization and composition approach that facilitates software requirements with several mechanisms at different levels of abstraction. It determines a level of composition with respect to the granularity defined in the component structure including atomic component (i.e., an elementary units of deployment, versioning, and replacement), composite component (i.e., a second-order level comprising a set of relevant components participating in a well-understood design pattern), and architectural component (i.e., a third-order level focusing on a component framework that constitutes a reusable design applicable to a given domain context). It provides a level of customization by embedding the coordination behaviors in the software architecture; for example, a coordination process that controls the business process and a client-server architectural style that utilizes the middleware infrastructures are aggregated into the coordination entity. CCM facilitates the separate concerns including computation and coordination. Furthermore, a policy that defines the overall strategy of the constituent software components is explicitly defined as the use-context policy (IPoEc) and role interaction policy (IPoIr). IPoEc is a policy that is imposed on the use-context and then realized by system policy services (e.g., ORB services such as TP monitor service, event service, and security service) at the ORB service level. IPoIr defines a policy service that is imposed on the interaction protocol (IP) and expresses interaction patterns and constraints enforced on the interaction among constituent participants, e.g., data encryption and fault-tolerance. Interaction protocol is the implementation of a coordination process and it may compose of a set of participating interaction rules (IRs), which is an abstraction unit of business actions that must be accomplished in order to comply with the specified interaction protocol for triggering actions and transforming data among components.
3. Role Model

A role is a modeling concept that is defined as an identifier for a behavior, which can be realized by a related software component. A role is an observable behavioral aspect of a collaborating software component (application object) with other interdependent components in order to fulfill required purposes (goals) while playing that particular role. There are three aspects of role abstraction that form three related role modeling based on the association with object instances in object orientation, the use of role model to capture patterns of interaction, and the organizational view approach.

The role model specifies the key role characteristics in the organizational structure including its responsibilities, permissions, relationship and constraint imposed by the policies as defined in the role meta-model (See Figure 2)

![Figure 1. Component Coordination Model.](image)

![Figure 2. Role meta-model.](image)

The role model composes with many role types that captured with role type specification template (Figure 3) toward the CCM. The role model precisely describes
all the roles that constitute the computational organization and their position in that organization. The role type are used for identifying various aspects of the large and complex enterprise systems in the term of permissions, responsibilities and relationship.

The following keys are used to identifying role type in the systems: Organizational positions and Organizational structure.

<table>
<thead>
<tr>
<th>Role Type:</th>
<th>a name of a given role in the role model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>a short description of the role and its characteristics.</td>
</tr>
<tr>
<td>Responsibilities:</td>
<td>the certain functionality and the obligations that a role has to perform including liveness and safety properties.</td>
</tr>
<tr>
<td>Liveness:</td>
<td>a set of protocols performed during a role life-cycle.</td>
</tr>
<tr>
<td>Safety:</td>
<td>- pre- and post-conditions.</td>
</tr>
<tr>
<td></td>
<td>- invariants.</td>
</tr>
<tr>
<td>Permissions:</td>
<td>define explicit policies associated with a given role and its context.</td>
</tr>
<tr>
<td></td>
<td>- a use-context policy imposed on the role responsibilities.</td>
</tr>
<tr>
<td></td>
<td>- a role interaction policy imposed on the inter-role communications.</td>
</tr>
<tr>
<td>Relationships:</td>
<td>- external relationship</td>
</tr>
<tr>
<td></td>
<td>- The relationship between use cases and the relevant role responsibilities is defined in the role type specification.</td>
</tr>
<tr>
<td></td>
<td>- The relationship between an actor and the role that the actor plays is defined in the role type specification.</td>
</tr>
<tr>
<td></td>
<td>- internal relationship</td>
</tr>
<tr>
<td></td>
<td>- The relationship between participating role types collaborating with one another in order to fulfill provided role responsibilities—the inter-role communications.</td>
</tr>
<tr>
<td></td>
<td>- The relationship between participating role types and relevant role model compositions—the context dependencies.</td>
</tr>
</tbody>
</table>

Figure 3 Role Type specification template

4. Integrating and Enhancing Role Model and CCM

A coordination entity (CE) is an architectural component class that represents a role type. From the role model, a role type is a specific interface presenting a particular aspect of the organizational role and an instance of a run-time software component that realizes this role type (or a collection of role types via multiple interfaces) is a coordination entity that plays role whose type is that particular role type. Roles are modeled in role hierarchies, which are static type hierarchies of all role types (Figure 4).
Coordination is a process of managing dependencies among activities, which are core functional entities of an application and are represented by software components. In our component coordination model (CCM), a coordination process is represented by an interaction protocol (IP), which manages dependencies by coordinating the relationships, patterns of interaction, and constraints specified by their associated dependency. In addition, a policy imposed on the interaction between participating components is represented by a role interaction policy (IPolr), which controls additional constraints (e.g., data encryption policy) enforced on a specified inter-component communication at a meta-level architecture. A coordination entity (CE) is a run-time software connector that has interaction protocols and role interaction policies embedded as the attributes. An interaction protocol can be implemented either by any visual programming language (e.g., JBuilder, Visual Basic, etc.) that provides IDL-aware client-server stubs, or by embedding a protocol component containing a script that governs sequence and pattern of interactions among participating CCM-based software components. A role interaction policy is embedded as a policy component that constrains the inter-component communications (component interactions behavior) and is realized at the meta-level. An interaction protocol in the CCM architecture may be implemented as a script stored in the repository, e.g., database. By managing coordination processes as pre-defined interaction protocols persisted in the repository, the design and implementation of software component interdependencies are then converted to a routine design problem that can be reused without the need for developers to write new coordination program to glue the components together. Similarly, by controlling a role interaction protocol at the meta-level implementation of additional policy components, the flexibility for customizing applications to a routine of policy components replacement is provided. The architecture of the model is illustrated in Figure 5
5. Realization in ORB Services

This model utilizes standard middleware infrastructure (e.g., COM+, EJB) that provide system-level services for the sake of integrity of and enterprise system’s data across multiple applications, system services transparency, improved application scalability, security so that simplified the programming of the systems.

6. Enhanced Integrated Methodology Model

After integrating and enhancing the model, the proposed methodology model is illustrated in Figure 6. First the systems requirements are modeled in use-context model, next the use case model are mapped to role model and then each role type are map to the CE (Coordination Entity) in the CMM, finally use the traditional object oriented technology to analyze and design the system in the object-oriented fashion.
7. Conclusion and future work

The contribution of this paper: it has provided a component software development guideline for develop component based application. The CCM has provided specifications for expressing the dependencies and policies management but lack of starting step from the problem statement that required by system analyst. By enhancing this model by integrating Role model and enhancing the CCM model to support it. Our proposed methodology can be implemented using popular middleware infrastructure such as the COM+ technology.

Future work is researching about scripting language technology for implementing coordination process and computation section of the enterprises systems and implementing a tested system such as E-Commerce system using COM+ middleware infrastructure and compare to another methodology by using the software matrix to measure sensitivity of the system in the case of changing of user requirements.
Reference


