Supporting Software Development With Roles

Haibin Zhu, *Senior Member, IEEE*, MengChu Zhou, *Fellow, IEEE*, and Pierre Seguin

***Abstract*—Software development tools are very important in software engineering. Although roles have been acknowledged and applied for many years in several areas related to software engineering, there is a lack of research on software development tools based on roles. Most significantly, there is no complete and consistent consideration of roles in all the phases of software development. Considering the increasing importance and applications of roles in software development, this paper intends to discuss the importance of roles in software engineering and that of role-based software development; review the literature relevant to role mechanisms in software engineering; propose and describe a role-based software process; and implement a prototype tool for developing complex software systems with the help of role mechanisms.**

***Index Terms*—Role, role concept, role mechanism, software engineering.**

I. Introduction

ROLES are important in our social lives. More than 2000 years ago, Confucius stated: "If names are not rectified, then language will not be in accord with truth. If language is not in accord with truth, then things cannot be accomplished" [Lun Yu: Zi Lu No. 13, Section 3]. In this saying, he emphasizes the importance of names, or implicit roles. He also said: "Let the king be a king, the minister be a minister, the father be a father, and the son be a son" [Lun Yu: Yan Hui No. 12, Section 11]. In this saying, he emphasizes the importance of positions, or more technically, roles. Shakespeare said: "All the world is a stage, and all the men and women merely players; they all have their exits and entrances; and one man in his time plays many parts. (As You Like It, Act II, Scene 7)." These sayings all signify the importance of roles in our social lives.

Role concepts have been applied widely in behavioral science, management, sociology, and psychology for many years. Roles are very useful in modeling the authority, responsibility, functions, and interactions associated with managerial positions within organizations. Early in 1982, Turoff and Hiltz [40] introduced roles into electronic journal system design. In their electric journal, they designed different role interfaces such as author, editor, reviewer, and reader. The design of these roles regulated the information and process flows among all the users of the system. They also specify roles in their electronic information exchange system (EIES) with subsets of primitive privileges including append, link, assign, and use operations [41]. Roles were taken as an important metaphor in the system. In the EIES, users dealt with roles but not individual privileges. With the primitive privileges, they composed roles such as indexer, organizer, and contributor.

Recently, more and more papers that apply roles have been published. Attention has been paid to roles in different areas relevant to systems such as modeling, software engineering, access control, system administration, agent systems, database systems, and collaborative systems [44]–[46]. Roles can help team members avoid being inundated by overwhelming information. Individuals in a team should have clear positions, and their roles should be related but not interfere with each other.

Although there is a common belief that roles are important concepts, until now, no consensus has been reached as to how roles should be represented and integrated as components into information systems. Specific meanings are assigned to the term "role" in different areas of research. The researchers then take their special understanding to support the basic ideas of their specific research topics or development projects. Many different role concepts have been proposed and applied in different areas, and discussed from different viewpoints. The actual situation of role applications in information systems can only be described as in chaos. There are no clear statements of what roles are in too many papers to list in the references. We encounter many difficulties in classifying the relationships of role concepts and mechanisms in different areas of application. There are few relations and inheritances among the role mechanisms applied in different studies and applications. Many early papers on roles from information systems referred to no research on roles in behavioral science. Roles seem to be phantoms. They are everywhere and every explanation seems reasonable, but it is difficult to grasp, identify, and completely specify them. Hence, there is still a need to clarify role concepts in order to support the design of information systems. In software engineering, roles are applied in different ways for different purposes, such as, roles as modeling mechanisms, interaction media, analysis and design tools, components for process models, and human resource management tools.

Having studied the literature in behavioral science including management, sociology and psychology, there would be benefits for the information-systems field in applying the role concept as developed in the behavioral sciences, because an information system is a virtual community that should truly simulate a real one.

Software development is a difficult and complex task. It is a typical collaborative activity that involves group organization,

Manuscript received November 2, 2005; revised May 2, 2006 and July 20, 2006. This work was supported by the IBM Eclipse Innovation Grant Funding and Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Funding of Canada. This paper was recommended by Guest Editor G. Cabri.

H. Zhu and P. Seguin is are with the Department of Computer Science and Mathematics, Nipissing University, North Bay, ON P1B 8L7, Canada (e-mail: haibinz@nipissingu.ca; seguin_pierre@yahoo.ca).

M. Zhou is with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: zhou@njit.edu).

Digital Object Identifier 10.1109/TSMCA.2006.883170

job dispatching and cooperation among group members. The members should fulfill their obligations and respect the rights of others in a software team.

In order for an information technology company to stay competitive, it is essential for its software teams to rapidly develop high-quality software, i.e., to deliver a software product within budget and by the deadline. Software engineering has been practiced and researched for 30 years to fulfill this goal; however, failure of software is still widespread. Many companies and enterprises experience the pain of software failure. Many of the reasons of software failure listed by [10] are related to people, task division, and project management. To manage tasks, organize people, and facilitate their collaboration in a software development team, we need to adopt the successful methodologies from social sciences, psychology, behavioral sciences, and organizational theories. Supporting cooperation in software processes is a critical task that has many facets and implications [3]. The concept of role is widely used in these disciplines.

Another key point in software engineering is "to divide and conquer." "How to divide" has been a major problem for software engineering for over 30 years. Different methodologies have been proposed such as component-based, object-oriented, and service-oriented ones. From our experience, the current situation is not satisfactory yet. We need to pursue new methodologies to overcome the problem and promote software development to a new stage. We believe role-based software development (RBSD) is one such new method, because roles can be applied in project management, system analysis, system design, and system deployment. With roles, task management, and distribution will be made easy.

This paper contributes in the following aspects.

1) Further establishes the importance of roles in all aspects of software engineering.
2) Proposes a process of role-based software engineering.
3) Develops a prototype tool to dynamically add, specify, and modify roles. Based on our research, we believe that these tasks are evidently required in all the phases of software development.

This paper is organized as follows. Section II reviews roles applied in different aspects of software development. Section III proposes a process of RBSD. Section IV describes the design and implementation of a software development platform prototype. Section V concludes this paper and indicates the further research topics.

II. Roles in Software Development Versus Role-Based Access Control (RBAC)

When we discuss role-based approaches, many people mention RBAC. RBAC was proposed early in the 1990s, and the term "role-based" was used mostly in access control and system management in the past 15 years. Because of the limitation of the view of access control, the role concepts and mechanisms in RBAC cannot transcend the requirement of access control, even though many people have tried to give a more abstract role concept and more generalized role model. The role characteristics in RBAC are as follows [5], [11], [13]–[15], [20], [27], [35].

1) Least Privilege: It requires that users be given no more privileges than necessary to perform their job function.
2) Separation of concerns: 1) A role can be associated with an operation of a business function only if the role is an authorized role for the subject and the role was not assigned previously to all of the other operations. 2) A user is authorized as a member of a role only if that role is not mutually exclusive with any of the other roles for which the user already possesses membership. 3) A subject can become active in a new role only if the proposed role is not mutually exclusive with any of the roles in which the subject is currently active.
3) Cardinality: The number of users taking a role cannot exceed its cardinality, i.e., the maximum number of users who can take this role.
4) Dependency constraints: There is a hierarchy or such relationships among roles as contain, exclude, and transfer.

The role concept in RBAC actually comes from the idea used in operating systems. A role is a tag that can be used by the system to perform protection on resources. In the UNIX operating system, every user is categorized as an owner, group member or one of others. These are actually roles for a user to access a file. The system grants users, based on their roles, access rights to files in order to accomplish protection. The major problem is that RBAC only considers the benefits obtained by assigning access permissions to users. This is a common view of software systems, i.e., a user is a client and the system is a server. RBAC emphasizes roles with respect to rights only.

In fact, in a social environment, roles are taken as a tool to specify human behavior. Their results are a good guide for our software developers to organize, analyze, and design software systems. In RBSD, roles are taken as interaction mechanisms to facilitate the interaction among team members; roles are taken as modeling mechanisms to create and design system frameworks; and roles are taken as software construction mechanisms to produce executable software code.

Collaborative activities in software development require both aspects of roles: rights and responsibilities. Collaboration among objects in a system also requires that an object know both these aspects. Also, RBSD requires that a role be dynamic. A user's binding to a role may change from one application to another and perhaps even during a single session within one application. Clearly, this is not captured by the operating system-level role concepts of RBAC. We will see many more characteristics of roles to be detailed in Section III that are not included in the roles of RBAC. Roles in RBAC only cover access rights of system users to system objects. In RBSD, roles are expected to cover rights, responsibilities, accessibilities, and collaboration methods.

III. Software Development Research Related to Roles

Software engineering involves many aspects or steps from management, modeling, design, and coding. Therefore, roles are introduced into software engineering in many different

ways. From the waterfall model of software development, we know that the software development life cycle (SDLC) starts from project plan, to system requirement analysis, to system design, to coding, and finally to maintenance [30]. Roles as abstract concepts and as tools can be applied in almost every aspect in the life cycle. We say "almost" because roles have not been fully developed in all the SDLC yet. Roles have been introduced into software engineering to mainly help analyze, design, and manage software systems. The key is how to support software development with roles.

In this section, we only summarize those publications relevant to SDLC. In the past, roles were mainly concerned with notation systems, object collaboration, reusable component design, and project management. Roles have been applied into different aspects of software engineering with different emphasis, as reviewed next.

A. Roles Relevant to Management

To form a successful organization, we need to have the members in the organization play different types of roles, such as task-oriented, relation-oriented, and self-oriented ones [18]. Unclear role specification creates dysfunctional ambiguity and conflict in an organization [7], and clear role definition and specification can help a person collaborate in a group [2]. This is also true in software development. There has been extensive research on role dynamics. It is concluded that the stress of an organization comes from the role conflicts and role ambiguity [7]. According to [1] and [22], role conflicts mean a situation in which an individual does not know how he or she should behave because the expectations of two or more other people differ. Role ambiguity means a situation in which an individual does not know exactly how he or she is expected to behave because the expectations are set forth vaguely and abstractly. Dynamic role allocation requires a stable basis in order to improve productivity and performance [1]. In the future, people may be expected to dynamically change their roles according to the needs of the group and the organization [6]. Therefore, there is a strong requirement for clear role specification and dynamic role allocation in collaboration.

The current workflow products for office automation declare that in a system, every role has simple and effective tasks. From the role definition, users can understand their duties and improve their work efficiency. These products show us the requirements of roles in normal office jobs that are a type of collaboration.

In 1993, Cain and Coplien proposed a role-based empirical process modeling environment [9]. They considered roles as building blocks of organizational structures. In their process model, roles are longstanding jobs within a process, usually intuitively recognized by its culture, for which relatively stable job descriptions may exist. They applied classes, responsibilities, and collaborators (CRC) cards to describe roles and distributed CRC cards to team members to play them. They observed that the role-based processes were satisfactory to the team members.

In 1996, in the SPADE-1 [3] environment, Bandinelli *et al.* introduced static roles such as project manager, system administrator, designer, and programmer in their environment. Users can perform operations on the global workspace according to their roles. They used process modeling languages to offer capabilities to describe roles, manual and automated procedures, interaction among users, process artifacts, and constraints. Because their major task is to provide a collaborative environment for software engineering, they fail to develop roles deeply enough. They just use a string such as "ProjManager" to express the role of a project manager. In other words, their roles are used to simply specify the tasks or operations of a user. There are no discussions about how to add new or modify existing roles in their SPADE-1 environment.

Sheard [37] described 12 roles involved in software engineering in 1996: Requirements Owner, System Designer, System Analyst, Validation/Verification Engineer, Logistics/Operations Engineer, Glue among Subsystems, Customer Interface, Technical Manager, Information Manager, Process Engineer, Coordinator, and Classified Ads System Engineering. She mainly described roles by illustrating relevant responsibilities or jobs. She also pointed out and discussed the problems related to roles in systems engineering such as role allocation, role combination, and role interaction. In her paper, no solutions have been proposed.

In 2004, Acuna and Juristo published their practice in applying role concepts intuitively in software project management [1]. They concentrate on human resource management in software development and believe that the theory of psychology helps software project management. They define roles as sets of responsibilities and capabilities required to carry out the activities of each subprocess. A capability defines the skills or attributes of a person. Their method is capability-oriented, and roles are taken as the elements of a software process. Their method supports only project management at the level of concepts and methodology but they provide no tools to facilitate such work.

In project management, roles are used to express different behaviors. Roles are used to support cooperation among team members by specifying tasks and operations [1], [9].

B. Roles Relevant to Design

Roles are a fundamental concept in modeling. Roles are entities that temporarily confine the behavior of objects. A role is considered as an abstraction and decomposition mechanism related to objects. Objects may play roles. When an object plays a role, it accepts messages and provides services (or fulfill responsibilities) related to its role. A role constitutes a part of an object's behavior (or responsibility) that is obtained by considering only the interactions of that role and hiding all other interactions. Roles are defined as a concept that is founded but not semantically rigid [17]. "Founded" means that a concept can exist essentially independently. "Semantically rigid" means that a concept contributes to the identity of its instance. This definition can help determine if a concept is a role. Roles allow not only for the representation of multiple views of the same phenomenon, but also for the representation of changes in time. Roles are also the bridge between different levels of detail in an ontological structure and for networking ontology of

different domains. The role analysis technique [12] is proposed to analyze dynamic programs. Its role concept reflects the important aspects of roles: the separation of concerns and role transitions. The working behavior of an object represents the specific context in which it is defined, together with other objects. All the actions in its working behavior belong to one or more of its roles.

In 1983, Holt pioneered research on role activity theory [19]. The theory was improved in 1995 by Ould who proposed role activity diagrams (RADs) to describe software processes [29]. RAD is generally used to show the responsibilities, drivers, and parallelism of the processes. A RAD comprises one or more role symbols annotated with role names. In RAD, a process is composed of roles. A role is composed of activities and taken as a means of associating human and other resources with tasks and processes.

In 2000, Murdoch and McDermid proposed a method to model engineering design processes with RAD [26]. They believe that various resources and events can be associated with a role. In their method, a process involves the concurrent activity of several roles at one level. RADRunner [34] is built as a tool to help use RAD to express business processes. This activates a new wave of applying roles in the process of analysis and design of software. The limitation of RAD is its lack of flexibility. Because a role specifies a part of a process, a role player must execute the process part as specified.

Roles are also used in order to encapsulate functionalities that may change dynamically when an object evolves. In the unified modeling language (UML), roles are taken as names of association ends, slots in collaborations, and dynamic classes. To provide appropriate mechanisms in UML for simulating roles, Steimann revised the UML's role concept in 2000 [39]. The diversity of concepts for UML to accommodate makes the UML diagrams difficult to understand including those expressed with role symbols as proposed. The new role concepts [39] have replaced the notions of association role and association end role as well as the rarely used association generalization.

In 1995, Reenskaug *et al.* applied roles to describe object-oriented software engineering processes [31]. A role in an object specification is called an object type that is a specification of a set of objects with identical externally visible properties. A role is a "why" abstraction. All objects that serve the same purpose in a structure of collaborating objects in a certain context are said to play the same role. They emphasize the role model but not the role itself and introduce roles intuitively. The separation of concern is the major consideration of their role model. They show a good application of role models in describing engineering processes. The references cited are mainly from object-oriented design, management, and psychology literature.

In 1996, VanHilst and Notkin introduced roles with both object collaboration and evolution in designing reusable components. In the collaboration view, a role is the part of an object that fulfills its responsibilities in collaboration [42]. Compared to classes, roles encapsulate fewer decisions and are thus more stable with respect to evolution. They provide a method to implement roles with C++ class templates and add roles into a class to refine or extend its interface. The composition and inheritance are used to express the collaboration between roles in their implementation. Their work again demonstrates that a class-based programming language can be expanded to support roles with evolution requirement.

In 1998, Riehle *et al.* introduced a role type to describe the view that one object holds on another object. A role type is described by using a type specification mechanism [32], [33]. A role object is one that represents one specific role of a core object to its clients. The role object wraps the central core object, which maintains its role objects for different clients. Collaboration is taken as a set of roles and their relationships. Hence, the design of a framework can be composed of role models, each of which may be a pattern instance. A role model actually describes a particular aspect of an object. Their roles facilitate the separation of concerns. It is clear that their work improves Reenskaug's [32] by providing the modeling roles that support framework design and integration.

In 1999, Kendall discussed a methodology to design role models with aspect-oriented programming [31]. She adopted and implemented the role concepts as modeling tools [32], [33]. According to her, roles are abstraction and decomposition mechanisms. A role can be specified in the language AspectJ. It is concluded that aspect-oriented programming is a promising approach to supporting role models.

In 2000, Bäumer *et al.* modeled context-specific views of an object with a role-object pattern as separate role objects. Role objects are dynamically attached to, and removed from the core object [4]. They separate two subclasses: core classes and role classes for an object class. They use subclasses of the role classes to express the roles played by the instances of the object class. Their method provides good guidance to design an object system with role concepts. It helps the role concepts be reused at the pattern level.

In the same year, Zhao and Kendall proposed the applications of role modeling in component design [43]. A role model is stated as an abstraction that describes the patterns of interactions among a set of entities. The entities play certain roles in a given context. The context is captured by the role model. A role model depicts frequently occurring but transient relationships among entities or objects that are working together to perform a certain task or accomplish a certain goal.

Although the aforementioned works have little in common, they indeed have some objectives, that is, to separate concerns, integrate rights and responsibilities, confine boundaries or specify processes. Roles are abstraction and decomposition mechanisms with notational systems in system analysis and design [19], [23], [24], [26], [29], [32], [34]. They are used as a tool to analyze the system requirements and ensure that future products reflect what a client hopes to receive. UML, as a formal diagrammatic tool [39], and RAD, as a formal process description tool [26], [29], [34], use roles to model groups of operations and processes. Roles are played by objects in software. In component design, roles are used to express frameworks for reuse [21], [43]. We have practiced applying roles in the constructions of role-based programs to express objects' evolution [25] and have demonstrated that software construction can be done by specifying roles,

TABLE I
ROLE MECHANISMS IN SOFTWARE DEVELOPMENT

Concerns	Year	Authors	Motivations	Contributions	Conclusions
Roles as modeling concepts	1983	Holt	To describe software process formally	Propose the role activity theory	Role activities can be intorduced into modeling an engineering process.
	1995	Ould	To continue Holt's work	Propose the notation systems for role activity diagrams	Roles are a good concept in notational tools to describe system processes formally.
	2000	Murdoch	To apply RAD into engineering process specification	Propose a method to model engineering design processes with RAD	RAD can be used to model engineering processes.
		Steimann	To clarify the role concepts in UML	Propose a revised UML metamodel building on a much simpler role definition	The metamodel introduces the interface concept while avoiding the inconsistence problems related to the roles in UML.
Roles as interaction media	1996	Bandinelli Di Nitto Fuggetta	To support cooperation among the team members in software development	Introduce static roles in a environment to support cooperative software engineering processes	Roles can be used to express software development processes and support cooperation among the team members.
Roles as analysis and design tools	1996	VanHilst Notkin	To improve code reuse and adaptation with roles	Propose a method to map role-based design to implementation in C++	Role-based design can be mapped to C++ Templates.
	1998	Riehle Gross	To use role modeling as an enabling technology for framework design and integration	Propose a role model-based framework for object-oriented software development	Using role models in designing frameworks makes clear the intent of the interaction between obejcts and between the frameworks and their clients.
	1999	Kendall	To implement role mechanisms	Propose a method to support role models with aspect-oriented programming	Aspect-oriented programming can support the implementation of the role model framework proposed by Riehle and Gross.
	2000	Zhao Kendall	To introduce roles to improve component design	Enable to identify components based on roles.	Roles can be considered as abstraction and representation of collaborative components.
Roles as components for process models	1993	Cain Coplien	To develop new process model with roles	Propose a method to guide task distributions to team members with roles	Team members are safisfied with the role-based processes.
Roles as human resource management tools	2004	Acuna Juristo	To improve the human resource management in software development	Propose a procedure to assign team members to roles in software project management	The role assigment procedure can make it easier to manage human resources as regards to role performance, thus to improve the productivity.

specifying objects, assigning roles to objects and having them play roles.

All the above discussions are relevant to the modeling and construction (including analysis, design, and coding) phases of software development [30].

C. Remarks

The aforementioned research demonstrates that it is possible to introduce roles into every activity of software engineering. Table I lists the relevant literature. All these topics remain to be explored and more practice is required. It is our hope that we create a consistent role model that can be applied over the whole software life cycle to positively and profoundly impact the software development practice.

In conventional software engineering, roles are applied in different ways for different purposes, e.g., roles as modeling mechanisms, interaction media, analysis and design tools, components for process models, and human resource management tools. The major problems of these applications are that they have not really understood the nature of roles and developed a practical mechanism for reusing, managing, and applying roles. In these applications, roles were still a very ambiguous and abstract concept. Also, in conventional software engineering, roles are considered as a mechanism to solve one aspect or one activity confined in software engineering, such as, management, analysis, modeling, and design. The roles and role mechanisms in conventional software engineering are scattered, loosely related, and inconsistent.

By whether roles have concrete specifications, we have the following.

1) Interface Roles: Roles are abstract entities to express the interfaces between objects, agents or people in collaboration. In this sense, roles only specify what their services are and what their requests are. How the services and requests are processed depends on the role players. We can call these interface roles.
2) Process Roles: Roles describe the concrete behavior of objects, agents, or humans. They specify not only services and requests but also how services and requests are processed. We can call these process roles.

IV. RBSD

In software development, there are communication, planning, modeling, construction, and deployment phases [30]. Software development is a process of mapping from a problem domain to its solution domain. Roles can be useful in all

the different phases from a generalized form in concepts to a concrete form in programming entities. It is beneficial for us to have a consistent view of roles.

The common meaning of the term "role" is derived from the theater and refers to the part played by an actor. A role represents a specific status that possesses certain rights and accompanying responsibilities. It can be defined as a set of regulations defining what the behavior of a position member should be. A role defines a set of responsibilities and capabilities needed to perform the activities relevant to these responsibilities [1]. A role is defined as the prescribed pattern of behavior expected of a person in a given situation by virtue of the person's position in that situation. It is simply defined as a position in a social structure [2]. A position means a more or less institutionalized or commonly expected and understood designation in a given social structure such as an accountant, mother, or church member [2]. A role is a set of expectations about the behavior of a particular position within a work system. Generally speaking, a role is a position occupied by a person in a social relationship. Within this position, the person possesses special rights and takes special responsibilities.

To discuss the properties of roles, we need at first to clarify some concepts of object orientation. An object is anything that forms an accessible entity in a software system. An object provides services by responding to the messages sent to it. An object can ask for services by sending messages to other objects. A component is an object or a group of objects controllable by a single person in software development.

Combining the above discussion with the review in Section III, we believe that a role presents two aspects: one is the services and the other is the right to ask for services [45]–[47]. From these two aspects, we could apply roles into every phase of software development.

1) A role is independent of players.
2) A role can be created, changed, and deleted.
3) A role includes both responsibilities when players are taken as a server and rights when they are taken as a client. To specify a role means to specify both responsibilities and rights.
4) As for the service interface, a role is actually a list of messages to be sent to an entity. As for the request interface, a role expresses or restricts the accessibility of an entity to the system.
5) Roles can be taken as a media for interactions. In communication, planning, and deployment phases, roles are described with written documents and are taken as working guidelines.
6) Roles are taken as interface specification tools. In modeling and analysis phases, roles are used to describe objects' jobs.
7) Roles are taken as process specification tools. In the design and coding phases, roles are described by concrete coding segments to confine the processes of objects.

In RBSD, a role is a tool for task management and a modeling mechanism for software component design. Roles are the basic and kernel concepts. Roles help one follow the principles and approaches for software development. They are clearly and exactly specified. In RBSD, roles and their supplementary mechanisms should be able to give a solution in a systematic way for problem solving with software. In RBSD, via roles, we will prevent developers from being overwhelmed by too much irrelevant information and tasks. To reach such a goal, the problems to be solved include what roles are, how to express, present, store, and change a role. Roles and role mechanisms in RBSD need to be integrated, consistent, and complete.

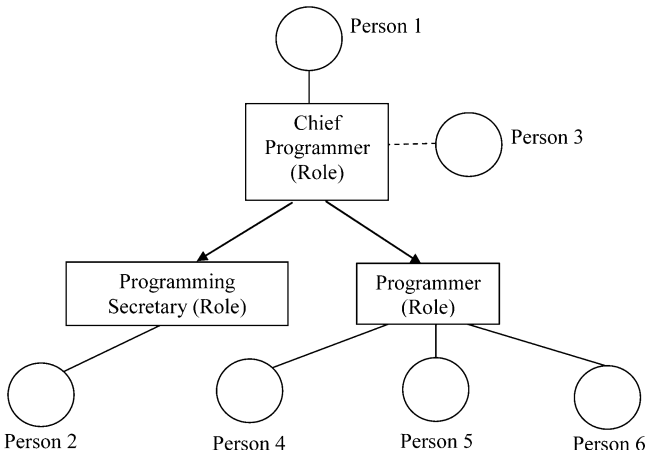


Fig. 1. Communications in a role-based team.

The benefit of roles in RBSD is that developers could complete their jobs within their allowed rights, i.e., visible contexts or objects. In traditional object-oriented software engineering, developers are granted rights to access all accessible objects (classes) in the system to be developed. This granted ability is the main source of poor readability and maintainability, i.e., to understand a class, one needs to understand all the classes (at least the interfaces of these classes) used in the class.

Roles can help in RBSD:

1) identify the human user "self";
2) avoid interruptions;
3) enforce independence by hiding people under roles;
4) encourage people or agents to contribute more;
5) remove ambiguities to overcome expectation conflicts;
6) work with specialized interfaces;
7) concentrate on a job and decrease possibilities of conflicts for shared resources;
8) transfer roles based on the requirements of a group;
9) separate concerns for complex components;
10) specify concrete processes relevant to specific requirements.

A. Activities of RBSD

Using a role-based approach, software development is composed of four activities: team management (Fig. 1), architecture design (Fig. 2), component design and implementation (Fig. 3), and system integration (Fig. 4). We use activities instead of phrases, because they may be, sequential, interleaved, or parallel in terms of time. In team management, roles are designed for and distributed to team members, i.e., people. In architecture design, roles are designed for objects. Role specification and role relationships are the major tasks for designers to describe;

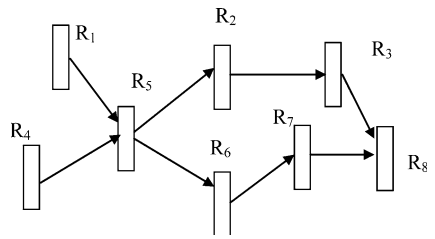


Fig. 2. Roles: Architecture design.

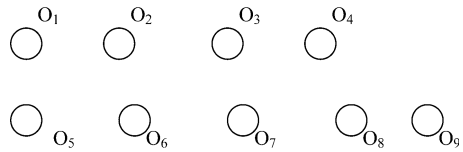


Fig. 3. Objects: Component design and implementation.

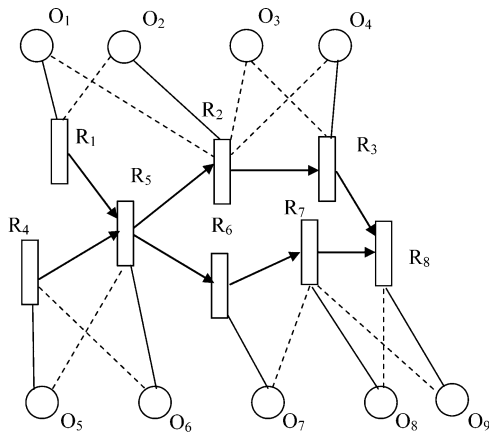


Fig. 4. Assign roles to objects: System integration.

in component design and implementation, object design and implementation are the major tasks for programmers; in system integration, objects, and roles are combined together to make a whole system executable. In all the activities of RBSD, we emphasize the following principles.

1) Clear role specification: It is easy for human users and modelers to understand their responsibilities and rights.
2) Flexible role transition: It is flexible and easy for a human user and components to transfer from one role to another.
3) Flexible role facilitation: It is easy for people, modelers, and programmers to specify roles. Because a system is developing, the existing roles might be required to adjust to correspond to the development of the system.
4) Flexible role negotiation: There are two aspects for role negotiation. One is that it is easy to negotiate the specification between a human user and a role facilitator. For example, a programmer may negotiate with the project manager for some special objects to develop. The other is that it is easy for people to negotiate roles for objects. For example, analysts, designers, and programmers may negotiate with each other about the roles of an object and some special aspects of a role.

In Figs. 1–4, a rectangle is used to express a role; a circle an object or a person; a solid arrow requests/services; a dashed line a role attachment which means the object (person) is able to play the role; and a solid line a current role which means that the object (person) is playing the role.

1) Team Management: We can use roles to make organization more efficient. A role-based team that derives from the chief programmer team [36] is shown in Fig. 1. In this team, a person only contacts or communicates with the roles specified in the team. Adding a new person into the team does not increase the communication cost among persons. It only adds one line of communication between the newly added person and the roles she/he is playing.

2) Architecture Design: Designers are mainly concerned with specifying roles and the relationships among roles. In design, roles are created and put into the pool (Fig. 2).

In architectural design, role specification and role relationships are the major tasks for designers to describe; in component design and implementation, object design and implementation are the major tasks for programmers; in system integration, objects and roles are combined together to make a whole system executable.

3) Component Design and Implementation: In RBSD, programmers are mainly concerned with implementing classes and objects that can play roles (Fig. 3). The tasks of object implementation are to accomplish the services to respond to the requests with special objects (classes with data structures and algorithms). In this implementation, programmers may work in parallel. Experienced and highly skilled programmers may produce more objects than novice programmers. Suppose that there are M objects, N programmers making objects and each programmer completes one object at T time units. The time to complete M objects is MT/N. Therefore, adding more (Δ) programmers will certainly shorten the time to complete the objects $MT(N+\Delta)$. This potentially breaks Brooks' law [8] which states that adding manpower to a late software project makes it later.

There are two possibilities for an object that cannot work: One is that the designers do not give it enough provided requests (or rights); or the coders do not completely apply the provided requests (rights). The designers should concentrate on specifying roles and the relevant requests and services, they need only care about the high-level provision and request relationships, that is, what roles should be in a system, what roles require, and what roles provide. The implementers mainly work on how to implement the services using the provided requests.

4) System Integration: With roles and objects, a software product can be obtained by integration, i.e., assigning roles to objects and having objects play roles. This step builds the deliverable software product (Fig. 4). At this step, objects are assigned to match a role or roles to play. If each role has a relevant object to play it and each object can provide the function, space, and speed requirements of that role, the system is complete.

Evidently, the above three steps can be done by specialists such as designers, programmers, and system integrators with different experiences and trainings. The integration step shows scaling capabilities based on how many objects play a single role. The efficiencies of different objects provided by different programmers may be different. Based on these differences, the managers of the development team have concrete evaluation criteria for programmers.

From Figs. 2–4, a system is designed when roles are designed and specified. System construction is to design objects that are qualified to play the roles and arrange objects to play roles.

With Figs. 2 and 3, software design is to design roles, and the relationships among the roles, and software implementation is to program objects and have the objects play roles.

B. Role-Based Processes

By a role-based software process, we mean that in software development there should be clear role specifications, flexible role transitions, flexible role facilitations, and flexible role negotiations as stated in Section IV-A.

From these properties, we find that roles are the key media for people to interact, collaborate, and model. The people are allowed to concentrate on their own roles. Role specification and negotiation are the major tasks in software engineering management. The specification of the interactions among roles actually defines the processes of management. In the sense of management, roles are played by people.

On the other hand, objects play roles from the viewpoint of programming, i.e., analysis, design, and construction. We can view role-based programming as an advance in programming methodologies. In each new methodology, the tasks of its predecessors are valid but become less important. We note the following three programming methodologies.

1) In procedural programming, composing procedures and specifying procedure calls are the most important tasks of programming.
2) In object-oriented programming, composing classes, instantiating objects, and specifying message passing are the most important tasks of programming.
3) In role-based programming, composing roles and specifying how objects play roles are the most important tasks of programming.

An RBSD process including management, analysis, design, and construction is as follows. Note that it is iterative for architecture design, object implementation, and system integration.

Step 1) Negotiate and specify roles: Team members discuss or negotiate to specify the roles relevant to people or components. If no compromise or agreement is obtained, abort the process.

Step 2) Assign roles: Every person or component is assigned one or more roles. If no agreement is obtained, abort the process.

Step 3) Play roles: People work according to their roles until the work is successfully completed or some conflicts or discontents occur.

Step 3.1) Provide services and do the work confined by the role: People understand what they need do at this time. Modelers, designers, and programmers understand how to describe the processes to serve. Incoming messages are confined by the role responsibilities (the service interface). If conflicts or discontents occur, the collaboration goes to Step 1).

Step 3.2) Request for help: To provide services, people need access to and interact with the environment by sending messages, or asking for resources and others' services. Modelers and designers need to describe components by playing roles and requesting other components. The messages are confined by the role rights (the request interface). If conflicts or discontents occur, the collaboration goes to Step 1).

Based on the above description, we can judge if a software system is role based or not. In fact, many traditional systems that apply role concepts cannot be called role-based systems because they support only some role views but do not use roles as the underlying mechanisms.

In a conventional software process model, ones have phases such as communication, planning, modeling, construction, and deployment. The role-based procedure [Steps 1)–3)] can be applied into every phase of the SDLC. The differences among the different phases are the role players. That is to say, in the modeling phase, the role players are software components such as classes; in the construction phase, they are objects; and in other phases, they are the team members.

C. Primitive Roles

Based on our E-CARGO model [46], we can obtain a new abstract software development team $\mathcal{T} ::= \langle \mathcal{C}, \mathcal{O}, \mathcal{A}, \mathcal{M}, \mathcal{R}, \mathcal{E}, \mathcal{G}, s_0, \mathcal{H} \rangle$, where $\mathcal{C}$ is a set of classes; $\mathcal{O}$ is a set of objects; $\mathcal{A}$ is a set of agents; $\mathcal{M}$ is a set of messages; $\mathcal{R}$ is a set of roles; $\mathcal{E}$ is a set of environments; $\mathcal{G}$ is a set of groups; s_0 is the initial state of the team; and $\mathcal{H}$ is a set of people.

A class c is a template for a group of similar objects. It describes the operations on the group of objects and specifies the data structure of these objects. $c ::= \langle n, \mathcal{D}, \mathcal{F}, \mathcal{X} \rangle$, where n is the identification of the class; $\mathcal{D}$ is a data structure description for storing the state of an object including pairs of classes and their external identifications; $\mathcal{F}$ is a set of the function definitions or implementations; and $\mathcal{X}$ is a unified interface of all the objects of this class.

An object o is everything in a system that occupies a memory cell or cells. It can be accessed and its data or status can be changed by operations on it. $o ::= \langle n, c, s \rangle$, where n is the identification of the object; c is the object's class identified by the class identification or name; and s is a data strucutre whose values are called attributes, properties, or states.

An agent a is a special object that represents a user involved in collaboration. It is defined as $a ::= \langle n, c_{\mathrm{a}}, s, \mathcal{N}_{\mathrm{r}}, \mathcal{N}_{\mathrm{g}} \rangle$, where c_{a} is a special class that describes the common properties of users, n is the identification or name of the agent, s is the set of properties of the agent, $\mathcal{N}_{\mathrm{r}}$ means a set of identifications of roles the agent is playing, and $\mathcal{N}_{\mathrm{g}}$ means a set of identifications of groups the agent belongs to.

A message m is defined as $m ::= \langle n, v, l, \mathcal{P}, t \rangle$ where n is the identification of the message, v is null or the receiver of the message expressed by an identification of a role, l is the pattern of a message, specifying the types, sequence, and number of

parameters, $\mathcal{P}$ is a set of objects taken as parameters with the message pattern l, where $\mathcal{P} \subset \mathcal{O}$, and t is a tag that expresses any, some, or all message.

A role r shows both the rights and responsibilities of human users. Incoming messages are used to express responsibilities and outgoing messages to express rights. It is defined as $r ::= \langle n, \mathcal{I}, \mathcal{N}_{\rm a}, \mathcal{N}_{\rm o} \rangle$ where, n is the identification of the role, $\mathcal{I} ::= \langle \mathcal{M}_{in}, \mathcal{M}_{\rm out} \rangle$ denotes a set of messages, wherein, $\mathcal{M}_{\rm in}$ expresses the incoming messages to the relevant agents, $\mathcal{M}_{\rm out}$ expresses a set of outgoing messages or message templates to roles, i.e., $\mathcal{M}_{\rm in}, \mathcal{M}_{\rm out} \subset \mathcal{M}, \mathcal{N}_{\rm a}$ is a set of identifications of agents that are playing this role; and $\mathcal{N}_{\rm o}$ is a set of identifications of objects including classes, environments, roles, and groups that can be accessed by the agents playing this role.

An environment e expresses a structure to build a group. It is specified by roles, the objects accessed by the roles and the cardinalities of agents playing roles. $e ::= \langle n, \mathcal{B} \rangle$, where n is the identification of the environment; and $\mathcal{B}$ is a set of tuples of role, number range and an object set, $\mathcal{B} = \{\langle n_{\rm r}, q, \mathcal{N}_{\rm o} \rangle\}$. The number range q tells how many users may play this role in this environment and q is expressed by [lower, upper].

A group g is a set of agents that work in an environment, i.e., a set of agents assigned with roles in the relevant environment. $g = \langle n, e, \mathcal{J} \rangle$, where n is the identification of the group; e is an environment for the group to work; and $\mathcal{J}$ is a set of tuples of identifications of an agent and role, i.e., $\mathcal{J} = \{\langle n_{\rm a}, n_{\rm r}, n_{\rm o} \rangle | \ \exists q, n_o (n_o \in \mathcal{N}_o) \wedge (\langle n_r, q, \mathcal{N}_o \rangle \in e \cdot \mathcal{B})\}$.

The initial state s_0 is expressed by initial values of all the components $\mathcal{C}$, $\mathcal{O}$, $\mathcal{A}$, $\mathcal{M}$, $\mathcal{R}$, $\mathcal{E}$, and $\mathcal{H}$ such as built-in classes, initial objects, initial agents, primitive roles, primitive messages, primitive environments, and initial personnel.

With the participation of people $\mathcal{H}$, $\mathcal{T}$ evolves, develops, and functions. Such participation includes joining a team $\mathcal{T}$, accessing objects of the team, sending messages through roles, and forming a group in an environment. The results of the team work are a new state of $\mathcal{T}$ that is expressed by the values of $\mathcal{C}$, $\mathcal{O}$, $\mathcal{A}$, $\mathcal{M}$, $\mathcal{E}$, $\mathcal{G}$, and $\mathcal{H}$, where $\langle \mathcal{C}, O, \mathcal{M}, \mathcal{R} \rangle$ forms the software product.

In a role-based team, people cooperating in software development comply with the rules in organizations relevant to roles. In a software development platform or a computer-aided software engineering tool, agents are special objects that represent people in a development team. Agents in software systems are more about autonomous objects. Objects and agents are assumed as the basic building blocks of software systems.

To support role-based processes, we need to provide primitive roles for a software development team. Based on the object principles, agent principles, role principles, and group principles [45] in role-based collaboration, the primitive roles should be as follows.

1) Class manager: This role allows a human user to create, delete, modify classes, or dispatch messages to a class. Newly changed classes are reloaded into the system dynamically.
2) Object manager: This role allows a human user to create, delete, and modify objects.
3) Agent manager: This role allows a human user to create, delete, and modify agents.
4) Environment manager: This role allows a human user to create, delete, and modify environments including adding or deleting objects and role ranges in an environment.
5) Group manager: This role allows a human user to create, delete, modify groups, assign roles to, or remove roles from agents in the group, or dispatch messages to a group.
6) Role facilitator: This role allows a human user to specify roles. To specify a role, the human user simply needs to drag and push the required messages or message patterns from a list of classes, agents, roles, groups, objects, and environments.
7) Role dispatcher: This role manages the messages sent to the roles it manages. The situation is as follows. Human users would like to ask for services. They search in the outgoing messages and find one. They then want to send the message. When they are asked to specify a receiver among agents, objects, and roles, they select a role. That means any human user who plays this role is fine to provide that service to them. Now, a human user who plays the role dispatcher will help to find a human user or agent to dispatch the message.
8) Role negotiator: This role allows a human user to approve another human user who hopes to play a role. The human user who plays the role negotiator manages the role to be played by another user. If people want to play a role, they must provide their agent class that implements all the required incoming message patterns.

D. Software Development Scenario

With the above primitive roles, we can define roles and apply our role-based process to software development. For simplicity of description, we can define roles such as project manager, analyst, designer, programmer, and tester. A special characteristic of our tool is that the users can add, modify, and delete roles dynamically. That is to say, the tool can accommodate any new roles required by the work. It is not restricted by the primitive roles in the tool. A scenario using this prototype to develop a software product is as follows.

1) With the group and role specification mechanism provided by the platform, we specify a development team with the roles of a project manager, an analyst, several designers, many programmers, and several testers and users in the platform.
2) A project manager can log into the platform with a role to get the interface specified by the project manager role. This role only concentrates on some special tasks of budget and time management, and some key technical issues of the project.
3) An analyst logs in with the interface of the role analyst and can accomplish the tasks relevant to him/her. His/her tasks include selecting models and tools, decomposing problems, distributing tasks to roles, and completing the requirement document and the analysis report.
4) A designer only views a subsystem of the project such as class hierarchy, class specification, and function specifications without caring about the code of every class and function.

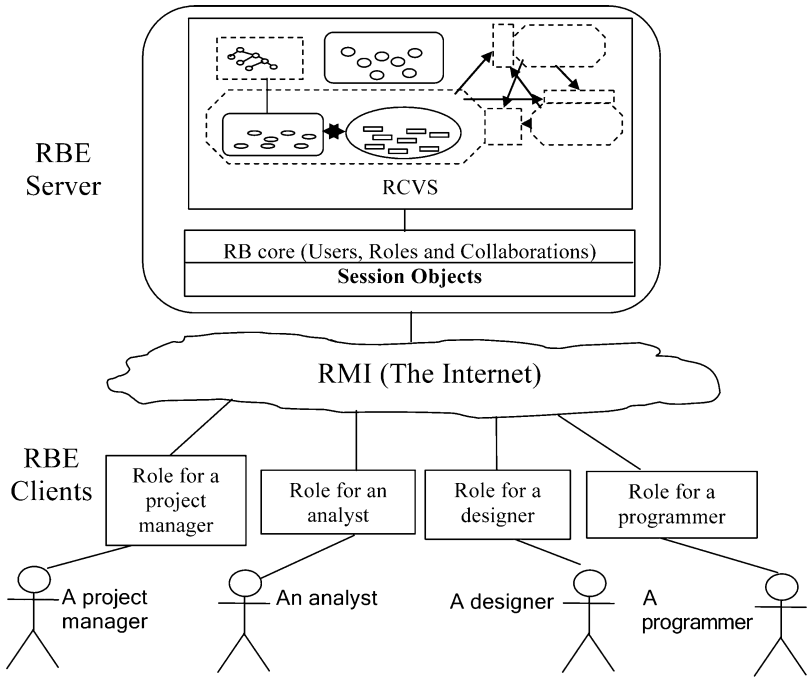


Fig. 5. Architecture of an RBSD platform.

5) Programmers view only their classes or functions and put their major efforts into coding the concrete classes and functions.
6) Testers can view and test the project in part or whole. They can submit testing results.
7) A user can try some objects and submit comments.
8) After programmers finish their jobs, they submit their code.
9) The designers collect all the completed parts into subsystems and submit them.
10) The analyst may collect all the subsystems and integrate them into one system, check testing results, collect comments from users, decide if there is a requirement to modify, and make the product deliverable.
11) The project manager manages budget and time, supervises, and coordinates with analysts and designers to guarantee the project is delivered on time and within the budget.

V. Prototype Tool for RBSD

Eclipse is a software development platform that supports programmers to make Java programs interactively with abundant programming tools. It is currently well received by Java programmers. As a general-purpose tool, Eclipse [28] can be used to develop tools and applications as diverse as web sites, embedded Java programs, C++ programs, and Enterprise JavaBeans. Although it provides a very good infrastructure that allows new plug-in software components to be added to the platform, it does not provide innovative and powerful mechanisms to support software project management and collaborations among software developers. Its major contributions are still in the provision of flexible low-level tools for programmers. To become a more competitive integrated development environment (IDE), it is very important for Eclipse to provide high-level tools for all the personnel relevant to the SDLC, such as users, testers, programmers, designers, analysts, and managers. Our aim is to extend Eclipse into a RBSD tool and extend it to support all phases of software development.

To construct a software team as discussed above, we compose an appropriate architecture based on the client/server model (Fig. 5). The clients support all the user interactions within the system:

1) class and object management interfaces;
2) role specification interfaces;
3) role playing interfaces;
4) role negotiation interfaces;
5) role transition interfaces.

The above interfaces are just frameworks to provide the interactions between users and the system. The details in the interfaces are determined by the roles played by users.

The server manages all the information of the system including classes, objects, agents, roles, environments, and groups.

- The database stores and manages all the permanent information of the system.
- The information exchange broker accepts and replies to all the requests from the clients.
- The tools help the role facilitators specify roles such as retrieving classes, objects, agents, roles, environments, and groups for special messages or message patterns.

The system is composed of a server and a set of clients (Fig. 5). The server is responsible for hosting collaborative projects, coordinating collaborative interactions, coordinating

the communication between clients, and deploying new types of collaboration. The server is composed of a role-based core (RB Core) and other collaborations that can be added as needed. In our prototype, the server is called role-based eclipse (RBE).

At the server side, RBE manages roles, user agents, and the installed collaborations. This kernel allows other components to interact with each other and access the mechanisms for dispatching, coordinating, and managing the rights of roles and users. With the help of the client/server structure and the deployment mechanism based on roles, the clients can transparently be updated with new collaborative components.

To show the usefulness of the plug-in facilities of the system and support a distributed collaborative software development environment, the RBE server also implements a role-based source control subsystem based on concurrent version system (CVS) called role-based CVS (RCVS).

To support the extensibility of our prototype system and to have it act as both an IDE and a rich client platform, we use Eclipse to support the implementation of our clients. The RBE clients communicate with the RBE server over remote method invocation to support real-time interactions.

A. RBE Server

The RBE server implements the E-CARGO framework discussed in Section IV-B. The RB Core subsystem manages roles $(\mathcal{R})$, agents $(\mathcal{A})$, and collaborations $(\mathcal{E}$ and $\mathcal{G})$. It allows clients to login, download the appropriate objects $(\mathcal{O})$ based on the classes $(\mathcal{C})$ in the server, and continue collaborations based on the downloaded objects. It has two main interfaces for a user to access: the user interface and the administrator interface. Each user can access the user interface by default so that they can interact with the system. The administrator interface is accessed through the default role of Administrator. Administrators have the ability to manage the system. They can manage roles by adding/removing roles or setting the access rights of roles. They can manage users by allowing or denying them access to the system. They can also assign roles to users and process role applications by users.

Agents are stored and managed at the server and are used to represent client users. Each user has a unique identity. Users can play roles, apply for new roles, and log into and out of the system. The agents are responsible for communication with users, keeping track of the roles that users are currently playing or can apply for, and supplying users with the proper objects based on their current roles. Roles also have rights in the form of access to interfaces. For example, the default role Administrator has access to the "Role Administration" interface of the system.

B. Collaborations

In our system, a collaboration is a component that is integrated into the system to extend it. A collaboration is actually a workable group $(g \in \mathcal{G})$ created based on an environment $(e \in \mathcal{E})$. Collaborations can be added to the system at startup or during runtime. Each collaboration in the system can interact with the kernel and/or other collaboration(s) already installed. Collaborations can either exploit the message-passing mechanisms of the roles or define new ways for the roles to collaborate based on their need. Collaborations define one or more perspective(s). Using the administration user interface, a role can be assigned access to one or more of the perspectives defined in a collaboration. Each perspective is associated with a session class. Objects of the session class exist on the server and are exposed to the client over a remote interface. The session objects can be queried for a client object that is passed to the client. Client objects interact with the session object and can do jobs that must be done on the client, e.g., displaying an appropriate user interface or downloading the files from the RCVS collaboration.

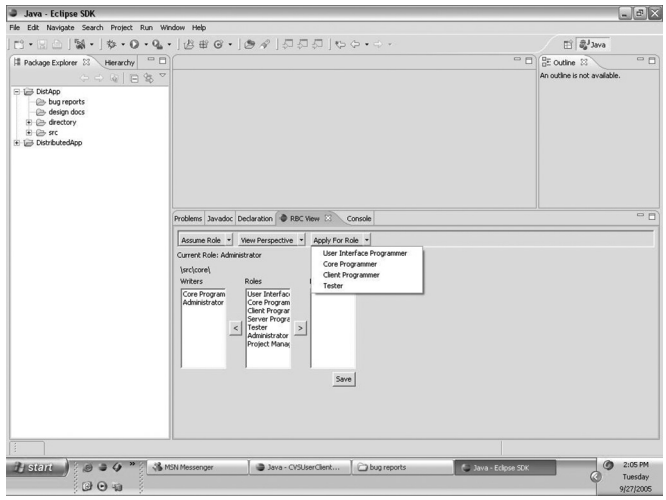


Fig. 6. Client user interface showing a user applying to act in a role. (Color version available online at http://ieeexplore.ieee.org.)

C. RBE Client

The client side of the system is implemented as a plug-in for Eclipse. The plug-in has mechanisms for connecting to the server, creating a new project on the server, or logging into a project that has already been created. Once the user is connected to the server the *user interface* is downloaded automatically and displayed in the RBC View window (Fig. 6). From this "root" interface the user can transition roles, apply for roles, and display other *interfaces* their current role has the access to. Fig. 6 shows the client user interface when applying for a role.

With the interfaces shown in Fig. 6, administrators can specify roles by setting the accessibilities to folders and files and other objects in a project. Similarly, programmers can add Java classes into the *src* directory, compile, debug, and test their own assigned software components such as classes. Designers, analysts, or project managers can obtain all the source code completed by the programmers and test a part or whole system. Users can also transfer to different roles easily to complete different tasks.

D. Role Playing Package

Role playing is the key mechanism in the implementation phase of an RBSD life cycle.

Roles can be taken as mechanisms to divide problems and facilitate changes. Traditional object-oriented programming methodologies take objects as unchanged entities after instantiations but in reality, objects evolve over time. We have practiced object and agent evolution with roles [25].

To support designing and coding software, we implemented a role playing package in Java. With this package, programmers can use Java to specify roles, define role players and have role players play roles in Java. The document of the package can be found at our website [47].

To define a role with our package, one only needs to define a class that extends class Role. To simulate an object playing roles, one only needs to define a class that extends class RolePlayer. With class Role, we can easily express the relationships among roles, i.e., the role relationships. When an object plays a role, it actually attaches an instance of a role class that is a subclass of Role. With classes RolePlayer and Role, we can easily express the relationships between an object and a set of roles. In the main class, we can easily simulate object evolution, role transition and role playing.

With the package we provide, the role-based process steps discussed in Section IV can also be used in designing and coding software systems. In this way, role specification and role assignment become the most important jobs in software design and coding. Role playing occurs when the software is running. This package also shows that the proposed role mechanism can be used to support object evolution and object collaboration.

VI. Conclusion

The contributions of this paper are as follows.

1) This paper further establishes the importance of roles in all aspects of software engineering.
2) It proposes a process of role-based software engineering.
3) It develops a prototype tool to dynamically add, specify, and modify roles. Based on our research, we believe that these tasks are evidently required in all the phases of software development.

RBSD is an exciting research area. Based on the RBSD platform, we may form a virtual software company by recruiting different staff for the development team. Programmers and designers from all over the world would have an opportunity to contribute to a software product. Roles have been applied in many areas for long. More attention must be paid to them in wider areas. For example, user interfaces present information in ways compatible with roles [16], [38]. Agents represent the roles of their human owners. In computer-supported cooperative work, people with roles can talk to others at a definite time. If a stranger calls them when they are busy with a special task, the current role would block such intervention. In management or workflow systems, one will do as the roles specify.

Software development is a systematic and complex task. We have only completed the first step toward a full RBSD methodology. There are still many challenges. We can foresee more interesting and successful activities in the research and practices of roles. In applying the role concept, a key problem is how to specify a role and how to apply a role in an organization or information system. The traditional role mechanisms and role concepts are not qualified to accomplish these tasks. Our proposed role mechanisms have solved some aspects of RBSD [44]–[46]:

1) integrate both the right and the responsibility aspects into roles;
2) unify the role concepts through the whole SDLC;
3) build roles as reusable components in different environments.

There are many essential topics that remain unsolved and require more comprehensive research:

1) improve the role mechanisms to allow easier specifications of messages, to match roles, and to deploy roles;
2) improve our prototype to support more activities in software engineering;
3) expand the collaboration activities with roles and to construct a real role-based collaborative system;
4) expand the prototype to support people to work by defining skills of roles, posting roles, accepting applications, evaluating them, and assigning roles to them;
5) evaluate the platform and assess its value.

Acknowledgment

The authors would like to thank L. Liu (Hunan University, China), for designing and writing programs to demonstrate the authors' ideas of role-based collaboration. His questions activated the authors' deeper thinking.

References

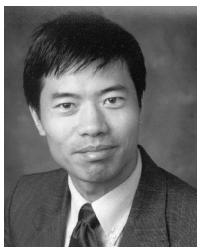
[1] S. T. Acuna and N. Juristo, "Assigning people to roles in software projects," *Softw. Pract. Exp.*, vol. 34, no. 7, pp. 675–696, Jun. 2004.
[2] B. E. Ashforth, *Role Transitions in Organizational Life: An Identity-Based Perspective*. Mahwah, NJ: Lawrence Erlbaum, 2001.
[3] S. Bandinelli, E. Di Nitto, and A. Fuggetta, "Supporting cooperation in the SPADE-1 environment," *IEEE Trans. Softw. Eng.*, vol. 22, no. 12, pp. 841–865, Dec. 1996.
[4] D. Bäumer, D. Riehle, W. Siberski, and M. Wulf, "Role object," in *Pattern Languages of Program Design 4*, N. Harrison, B. Foote, and H. Rohnert, Eds. Reading, MA: Addison-Wesley, 2000, pp. 15–32.
[5] E. Bertino, P. A. Bonatti, and E. Ferrari, "TRBAC: A temporal role-based access control model," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 191–223, Aug. 2001.
[6] B. J. Biddle and E. J. Thomas, Eds., *Role Theory: Concepts and Research*, Hoboken, NJ: Wiley, 1966.
[7] R. P. Bostrom, "Role conflict and ambiguity: Critical variables in the MIS user-designer relationship," in *Proc. 17th Annu. Comput. Pers. Res. Conf.*, Miami, FL, 1980, pp. 88–115.
[8] F. P. Brooks, Jr., *The Mythical Man-Month*. Reading, MA: Addison-Wesley, 1995.
[9] B. G. Cain and J. O. Coplien, "A role-based empirical process modeling environment," in *Proc. 2nd Int. Conf. Softw. Process*, Berlin, Germany, Feb. 1993, pp. 125–133.
[10] R. N. Charette, "Why software fails," *IEEE Spectr.*, vol. 42, no. 9, pp. 42–49, Sep. 2005.
[11] M. J. Covington, M. J. Moyer, and M. Ahamad. (2000). "Generalized role-based access control for securing future applications." in *Proc. 23rd Nat. Inf. Syst. Secur. Conf.* [Online]. Available: http://crc.nist.gov/nissc/2000/proceedings/toc.pdf

[12] B. Demsky and M. Rinard, "Role-based exploration of object-oriented programs," in *Proc. 24th ICSE*, Orlando, FL, May 19–25, 2002, pp. 313–324.
[13] D. F. Ferraiolo and D. R. Kuhn, "Role-based access control," in *Proc. NIST-NSA Nat. (USA) Comput. Secur. Conf.*, 1992, pp. 554–563.
[14] D. F. Ferraiolo, J. A. Cugini, and D. R. Kuhn, "Role-based access control (RBAC): Features and motivations," in *Proc. 11th Annu. CSAC*, 1995, pp. 241–248.
[15] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard: Role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 2, pp. 224–274, Aug. 2001.
[16] S. Greenberg, "Personalizable groupware: Accommodating individual roles and group differences," in *Proc. ECSCW*, Amsterdam, The Netherlands, Sep. 1991, pp. 17–32.
[17] N. Guarino, "Concepts, attributes and arbitrary relations: Some linguistic and ontological criteria for structuring knowledge bases," *Data Knowl. Eng.*, vol. 8, no. 3, pp. 249–261, Jul. 1992.
[18] L. R. Hoffmann, "Applying experimental research on group problem solving to organizations," *J. Appl. Behav. Sci.*, vol. 15, no. 3 pp. 375–391, 1979.
[19] A. Holt, H. R. Ramsey, and J. D. Grimes, "Coordination system technology as the basis for a programming environment," *Electr. Commun.*, vol. 57, no. 4, pp. 307–314, 1983.
[20] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "Generalized temporal role based access control model," *IEEE Trans. Knowl. Data Eng.*, vol. 7, no. 1, pp. 4–23, Jan. 2005.
[21] E. A. Kendall, "Role model designs and implementations with aspect oriented programming," in *Proc. ACM Conf. OOPSLA*, Denver, CO, Nov. 1999, pp. 353–369.
[22] R. C. King and V. Sethi, "The impact of socialization on the roles adjustment of information systems professionals," *J. Manage. Inf. Syst.*, vol. 15, no. 4, pp. 195–217, Spring 1998.
[23] B. B. Kristensen, "Object-oriented modeling with roles," in *Proc. 2nd Int. Conf. OOIS*, Dublin, Ireland, 1995, pp. 57–71.
[24] B. B. Kristensen and K. Østerbye, "Roles: Conceptual abstraction theory and practical language issues," *Theory Pract. Object Syst. Subjectivity Object-Oriented Syst.*, vol. 2, no. 3, pp. 143–160, 1996.
[25] L. Liu and H. Zhu, "Implementing agent evolution with roles in collaborative systems," in *Proc. Int. Conf. Netw., Sens., and Control*, Ft. Lauderdale, FL, Apr. 2006, pp. 819–824.
[26] J. Murdoch and J. A. McDermid, "Modeling engineering design process with role activity diagrams," *Trans. Soc. Des. Process Sci.*, vol. 4, no. 2, pp. 45–65, Jun. 2000.
[27] M. Nyanchama and S. Osborn, "The role graph model and conflict of interest," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 1, pp. 3–33, 1999.
[28] Object Technology International, Inc. (2003, Feb.). *Eclipse Platform Technical Overview*. [Online]. Available: http://www.eclipse.org/articles/index.html
[29] M. A. Ould, *Business Processes: Modeling and Analysis for Re-Engineering and Improvement*. Hoboken, NJ: Wiley, 1995.
[30] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed. New York: McGraw-Hill, 2005.
[31] T. Reenskaug, O. A. Lehne, and P. Wold, *Working With Objects: The OOram Software Engineering Method*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
[32] D. Riehle and T. Gross, "Role model based framework design and integration," *ACM SIGPLAN Notices*, vol. 33, no. 10, pp. 117–133, Oct. 1998.
[33] D. Riehle, R. Brudermann, T. Gross, and K. U. Mätzel, "Pattern density and role modeling of an object transport service," *ACM Comput. Surv.*, vol. 32, no. 1, pp. 1–6, Mar. 2000.
[34] Role Modellers Ltd. (2004). *A Better Way to Support Collaboration*. [Online]. Available: http://www.rolemodellers.com
[35] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *IEEE Comput.*, vol. 29, no. 2, pp. 38–47, Feb. 1996.
[36] S. R. Schach, *Object-Oriented and Classical Software Engineering*, 6th ed. New York: McGraw-Hill, 2005.
[37] S. A. Sheard. (1996). "The value of twelve systems engineering roles." in *Proc. INCOSE 6th Annu. Int. Symp.*, Boston, MA. [Online]. Available: http://www.incose.org/sfbac/welcome/12-roles.pdf
[38] B. Shneiderman and C. Plaisant, "The future of graphic user interfaces: Personal role managers," in *Proc. People and Comput. IX, Brit. Comput. Soc. HCI*, Glasgow, U.K., Aug. 1994, pp. 3–8.
[39] F. Steimann, "A radical revision of UML's role concepts," in *Proc. UML*, 2000, pp. 194–209.
[40] M. Turoff and S. R. Hiltz, "The electronic journal: A progress report," *J. Amer. Soc. Inf. Sci.*, vol. 33, no. 4, pp. 195–202, Jul. 1982.
[41] M. Turoff, "Computer mediated communication requirements for group support," *J. Organ. Comput.*, vol. 1, no. 1, pp. 85–113, 1991.
[42] M. VanHilst and D. Notkin, "Using role components to implement collaboration-based designs," in *Proc. ACM Conf. OOPSLA*, San Jose, CA, 1996, pp. 359–369.
[43] L. Zhao and E. Kendall, "Role modeling for component design," in *Proc. 33rd Hawaii Int. Conf. Syst. Sci.*, 2000, p. 8048.
[44] H. Zhu, "A role agent model for collaborative systems," in *Proc. Int. Conf. Inf. and Knowl. Eng.*, Jun. 2003, pp. 438–444.
[45] ——, "The role mechanism in collaborative systems," *Int. J. Prod. Res.*, vol. 44, no. 1, pp. 181–193, Jan. 2006.
[46] H. Zhu and M. C. Zhou, "Role-based collaborations and their kernel mechanisms," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 36, no. 4, pp. 578–589, Jul. 2006.
[47] H. Zhu. (2005). *Role Playing Java Documents*. [Online]. Available: http://www.nipissingu.ca/faculty/haibinz/RolePlaying

Haibin Zhu (M'02–SM'04) received the B.S. degree in computer engineering from the Institute of Engineering and Technology, China, in 1983, and the M.S. and Ph.D. degrees in computer science from the National University of Defense Technology (NUDT), Hunan, China, in 1988 and 1997, respectively.

He is an Associate Professor in the Department of Computer Science and Mathematics, Nipissing University, North Bay, ON, Canada. He was a Visiting Professor and a Special Lecturer in the College of Computing Sciences, New Jersey Institute of Technology, in 1999–2002, and a Lecturer, Associate Professor, and Full Professor at NUDT, in 1988–2000. He has published more than 50 papers, four books, and one book chapter on object-oriented programming, distributed/collaborative systems, and computer architecture. He was a Guest Associate Editor for the special issue of "Computer and Information Technology" for the *International Journal of Pervasive Computing and Communications*.

Dr. Zhu is serving and served as a Cochair of the technical committee of Distributed Intelligent Systems of the IEEE SMC Society, a Guest Editor for the special issue of "Collaboration Support Systems" for the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A, a program committee member for the 2006, 2005, 2004, and 2003 IEEE International Conference on SMC, 2006 IEEE International Conference on Networking, Sensing and Control, 2006 IEEE International Conference on Cognitive Informatics, 2006 IEEE International Conference on Services Computing, 2006 IEEE International EDOC Conference, and the 2004 Canadian Conference on Computer and Software Engineering Education (C3SEE'04). He is the recipient of the Best Paper Award from the 11th Illinois Society of Professional Engineers International Conference on Concurrent Engineering (ISPE/CE2004), the 2004 and 2005 IBM Eclipse Innovation Grant Awards, the Educator's Fellowship of Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)'03, a 2nd Class Nation-Level Award of Education Achievement from the Ministry of Education of China in 1997, a 2nd Class Nation-Level Award of Excellent Textbook from the Ministry of Education of China (2002), three 1st Class Ministry-level Research Achievement Awards from the Commission of Science Technology and Industry for National Defense of China in 1997, 1994, and 1991, and a 2nd Class Excellent Textbook Award of the Ministry of Electronics Industry of China (1996). He is a member of the Association for Computing Machinery and a Life Member of the Chinese Association for Science and Technology.



MengChu Zhou (S'88–M'90–SM'93–F'03) received the B.S. degree from the Nanjing University of Science and Technology, Nanjing, China, in 1983, the M.S. degree from the Beijing Institute of Technology, Beijing, China, in 1986, and the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1990.

He joined the New Jersey Institute of Technology (NJIT), Newark, in 1990, and is currently a Professor of electrical and computer engineering and the Director of the Discrete-Event Systems Laboratory. His research interests are in computer-integrated systems, Petri nets, wireless *ad hoc*, and sensor networks, system security, semiconductor manufacturing, and embedded control. He has over 200 publications including six books, over 90 journal papers, and 15 book chapters. He coauthored with F. DiCesare *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems* (Kluwer Academic, 1993), edited *Petri Nets in Flexible and Agile Automation*, (Kluwer Academic, 1995), coauthored with K. Venkatesh *Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach*, (World Scientific, 1998), coedited with M. P. Fanti, *Deadlock Resolution in Computer-Integrated Systems*, (Marcel Dekker, 2005), and coauthored with H. Zhu, *Object-Oriented Programming in C++: A Project-based Approach*, (Tsinghua University Press, 2005). He has led or participated in 30 research and education projects with a total budget over $10 million funded by the National Science Foundation, Department of Defense, Engineering Foundation, New Jersey Science and Technology Commission, and industry. He is Editor-in-Chief of *International Journal of Intelligent Control and Systems*.

Dr. Zhou was invited to lecture in Australia, Canada, China, France, Germany, Hong Kong, Italy, Japan, Korea, Mexico, Taiwan, and the U.S. He served as Associate Editor of the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION from 1997 to 2000, and currently Managing Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C and Associate Editor of the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING. He was General Cochair of the 2003 IEEE International Conference on Systems, Man, and Cybernetics, Washington DC, October 5–8, 2003, and Founding General Cochair of the 2004 IEEE International Conference on Networking, Sensors, and Control, Taipei, March 21–23, 2004. He organized and chaired over 70 technical sessions and served on program committees for many conferences. He was Program Chair of the 1998 and 2001 IEEE International Conference on Systems, Man, and Cybernetics (SMC) and the 1997 IEEE International Conference on Emerging Technologies and Factory Automation, and a Guest Editor for the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS and IEEE TRANSACTIONS ON SEMICONDUCTOR MANUFACTURING. He was General Chair of the 2006 IEEE International Conference on Networking, Sensors, and Control, Ft. Lauderdale, FL, in April 23–25, 2006. He was the recipient of a National Science Foundation Research Initiation Award, CIM University-LEAD Award by the Society of Manufacturing Engineers, Perlis Research Award by NJIT, Humboldt Research Award for U.S. Senior Scientists, Leadership Award and Academic Achievement Award by the Chinese Association for Science and Technology-USA, Asian American Achievement Award by the Asian American Heritage Council of New Jersey, and Distinguished Lecturer of the IEEE SMC Society. He was the Founding Chair of the Discrete Event Systems Technical Committee of the IEEE SMC Society, and Cochair (founding) of the Semiconductor Factory Automation Technical Committee of the IEEE Robotics and Automation Society. He is a Life Member of the Chinese Association for Science and Technology-USA and served as its President in 1999.

Pierre Seguin is an undergraduate student at Nipissing University, North Bay, ON, Canada.

His research interests are in software development, JAVA programming, and role-based collaboration.