Abstract—Computer-supported cooperative work (CSCW) systems are computer-based tools that support the collaborative activities of human users. They should not only support virtual face-to-face collaborative environments but also improve face-to-face collaboration by providing more mechanisms to overcome the drawbacks of usual face-to-face collaboration. Introducing roles into CSCW systems is important in achieving this. This paper’s contributions include establishing the requirements for a role-based collaboration, presenting the concept, requirements, and principles of role-based collaboration, proposing a model E-CARGO for role-based collaboration, and describing the kernel mechanisms and their implementation to facilitate the development of role-based collaborative systems for industrial applications.

Index Terms—Collaboration, computer-supported cooperative work (CSCW), role.

I. INTRODUCTION

How to build more flexible and practical collaborative systems is clearly an important research topic. Collaboration technologies have the potential to enhance the effectiveness of teamwork within and between organizations. The roles played by the participants in a collaborative activity are important factors in achieving successful outcomes [36]. The role concept should be a key concern in the development of computer-based systems [28]. The contributions of [29] and [30] demonstrate the importance of roles in a collaborative system.

Some traditional computer-supported cooperative work (CSCW) systems, or groupware, have indeed applied the concept of roles [2], [6], [10], [14], [18], [29], [30], [32]. However, these systems apply the role concept intuitively without defining it clearly and exactly. Smith et al. [30] state that people in a group play various roles and those roles are in general supported by a system’s treatment of output and user input. Singley et al. [29] assign many roles to foster users’ interactions in their collaborative learning environment. In electronic information exchange system (EIES), roles are built out of a subset of the primitive privileges (such as append, link, assign, and use) that are crucial to the human communication process [32]. In [18], there are roles for writers, readers, and commentators. Guzdial et al. [14] recognize the usefulness of roles in a collaborative tool CoWeb in which roles are specific concerns and activities associated with users. In CoWeb, many concrete roles are specified such as purpose agents, site designers, developers, central users, peripheral users, administrators, and support staffs.

Edwards [10] introduces access control policies and roles to avoid chaos in collaborative applications. The term role in role-based access control (RBAC) is used in assigning different access rights to categories of users. All users in a certain role share a set of access control rights to objects within the application. However, this concept of role does not involve a specification of collaborative working.

As a summary, there seems to be no commonly accepted concept of roles in traditional CSCW systems; there are no definite role specification mechanisms that are dynamically changeable in a collaborative system.

As for role-based collaboration, there are sharply different opinions.

- It is meaningless because collaboration itself implicates role assignments and role specifications.
- Roles have been introduced into information systems for over 20 years and all the problems have been solved.
- The software with roles is very restrictive [32]. It is neither welcome nor useful.
- Roles are not encoded solely in human biology or physical law [30]. It is almost impossible to describe what they are.

Therefore, the first task is to clarify the fundamental concepts of roles and related mechanisms. This paper responds to a perceived need to clarify the motivation for role-based modeling, the fundamental concept and definition of a role in the CSCW context, and the information system mechanisms needed to implement role-based CSCW systems.

This paper is arranged as follows. Section II establishes the requirements for the role-based CSCW systems. Section III presents the fundamental concepts of roles and role-based collaboration. Section IV describes a formal model for it. Section V discusses the architecture and components of role-based collaborative systems. Section VI implements some required kernel mechanisms in the proposed systems, and Section VII concludes this paper.

II. REQUIREMENTS FOR ROLE-BASED COLLABORATION

Collaboration is a common activity in a society of people. It means to work together [16]. Collaboration or group work is required and essential when one single person cannot complete a whole task. CSCW is receiving increased attention due to competitive pressures on companies to improve product development and other decision-making processes and to use information and communication technology (including the Internet technologies) as effectively as possible. CSCW seeks to enhance collaboration among people. However, the question of how to maximize productive collaborations by
manipulating role assignments and the configuration of teams is still largely unexplored [29]. Role-based collaboration is a recent innovation that addresses these concerns. It is a new methodology to organize collaboration by providing role specification, assignment, transition, and negotiation mechanisms. With these mechanisms, people in collaboration are clear about their roles, thereby making collaboration more productive.

Based on the discoveries and requirements of managerial, social, and psychological science [1], [4], [5], [16], a role-based collaborative system allows users to enhance the collaboration by negotiating, tuning, and transferring relevant roles in the system. The proposed role-based collaboration and theory can gain significant applications in group decision support systems (GDS) [5], [32], [36], management [27], [34], personalized interfaces [12], [25], [28], [30], agent systems [2], [7], and software engineering [8], [20], [22], [26], [33].

Collaboration requires that participants fulfill their obligations and respect the rights of others. It requires that everybody respects the social laws of the community [2]. Therefore, individuals fill specific positions and play specific roles in an organization. To collaborate, people generally join a group or organization. All individuals should have clear positions within a group and their roles should be related but not interfere with each other.

Unclear role specification may create dysfunctional ambiguity and conflict in an organization [5], while clear role definition and specification can help people collaborate in a group [3]. There has been extensive research on role dynamics. Role conflict and ambiguity are causes of stress in organizations. Role conflict means situations in which individuals do not know how they should behave because of their different expectations. Role ambiguity means a situation in which individuals do not know exactly how they are expected to behave because of vaguely and abstractly set expectations [1], [19]. Singley et al. [29] state that a collaborative learning environment may have to establish and assign well-defined roles to participants to foster interaction. Dynamic role assignment requires a stable basis to improve productivity and performance. By “dynamic” we mean that role assignment and reallocation occur during collaboration. People may be expected to dynamically change their roles according to the needs of the group and organization [8], [22]. Even though some argue that there are benefits from having “fuzzy” role definitions, particularly for more creative or uncertain work areas, there is still a period when the collaborators keep their definite roles and the “fuzzy” state should be resolved by role transitions before operations begin. Therefore, roles are required to be taken as underlying mechanisms in collaboration. The current workflow products for office automation declare that, in a system, every role has simple and effective tasks. These products show us that there is the same requirement of roles in normal office work that is a type of collaboration.

The requirements for role-based collaboration can be developed in the context of typical objectives for the CSCW systems. These may be considered as follows.

1. **Information sharing**: Collaborators require that information be shared to complete their common tasks. This sharing includes data sharing, such as shared documents, and view sharing, such as common visual information, for all the collaborators.

2. **Communication**: Collaborators require that information be exchanged efficiently and at a high speed over the network on which a CSCW system is built.

3. **User satisfaction**: Collaborators should be satisfied with the results obtained by their collaboration on a CSCW system. With roles, people can collaborate without contacting directly with people if they do not want.

4. **More productive collaboration and better collaboration performance**: With a CSCW system, the collaborators should be more productive and perform better than they do in a natural environment without it. Roles are such candidate mechanisms, because they can be used to avoid or reduce conflicts and ambiguity. The productivity can also be improved through better coordination of work as in workflow systems.

5. **Cooperation encouragement**: Collaborators should get continuous encouragement while participating in collaboration via a CSCW system. Roles can be taken as such a mechanism, because roles can be taken as ranks and flags of people’s success in collaboration.

6. **Intelligent help**: Collaborators should get assistance once in their collaborative activities. Agents as representatives in a system may help people respond to simple requests and record requests. Agents and roles may help the collaborators to clarify their rights and responsibilities.

Role-based collaboration will contribute to both the above aspects with concentration on usability. It is composed of a set of concepts, principles, processes, components, and structures.

In summary, the role concept appears in many CSCW and related fields. It is not always well defined but seems to provide an important view that could bridge between user needs, human work patterns, system functionalities, and designs. There is a need for improved methods and technologies to support the specification, design, and implementation of role-based collaborative systems.

### III. Basic Concepts

#### A. Role in Collaboration

According to the Oxford English Dictionary (OED) [21], a role is
“the part or character which one has to play, undertakes, or assumes;”
“the part played by a person in society or life;” or
“the typical or characteristic function performed by someone or something.”

OED [21] also defines a role in a behavioral and psychological view. It is “the behavior that an individual feels it appropriate to assume in adapting to any form of social interaction; the behavior considered appropriate to the interaction demanded by a particular kind of work or social position.”

In common usage, the term “role” derives from the theater and refers to the part played by an actor. A position represents a specific “seat” that entails certain privileges and accompanying responsibilities [5]. It is defined by Thomas and Biddle [4] as a set of prescriptions defining what the behavior of a position member should be.

A role is defined as a prescribed pattern of behavioral expectation of a person in a given situation by virtue of the person’s position in that situation [15]. A position means a more or less institutionalized or commonly expected and understood designation in a given social structure such as accountant, mother, and church member [1]. Bostrom [5] defines a role as a set of expectations about behavior for a particular position within a work system. A role is a position occupied by a person in a social relationship. At this position, the person possesses special rights and takes special responsibilities.

In business management, role modeling (RM) is proposed as a business engineering technique to provide a model of an organization in terms of roles, responsibilities, and collaboration among individuals and teams, and a discovery and transformation process for an enterprise, applicable in small or large scale and a vehicle for reengineering and process improvement [23], [27].

Generally, we can describe rights and responsibilities in natural languages. However, it is hard to describe roles clearly and strictly because they are ambiguous to some extent. That is why different people in the same position can make different contributions. In an information or software system, we may specify roles clearly with some special considerations. The possibility comes from the exactness of computer languages, limited types and number of resources a person can access at a time, and automatic tools to retrieve information relevant to the roles.

Information systems are developed today largely using object-oriented methods and technologies. Role concepts are widely discussed and used in object-oriented systems [3], [9], [11], [17], [20], [23], [24], [26], [31], [34]. Roles are object-dependent entities that are used to confine the behavior of objects temporarily, i.e., objects play roles. However, in collaborative systems, human users play roles. A collaborative system should pay more attention to human factors. The roles in collaborative systems should be good helpers for human users to collaborate. This motivates us to define the concept of role based on the role concepts used in behavioral science, sociology, psychology, and business management.

In collaboration, a person X faces an environment and other people who are involved. To collaborate with other people, X must know what they can do and would like to do. To work efficiently, X must know what objects can be accessed in the environment. To be cooperative, X must tell others what X can do and what X would like to do for them. Efficient collaboration means playing roles well. Therefore, in general, a person X has two kinds of existence: One is a server and the other a client. When X plays a role, X provides specific services and possesses specific rights to ask for services. With this common sense, a role can be defined as a person’s view of the collaborative environment. When people play a specific role, they have a specific view of the surroundings. Their role in a collaborative environment is actually a wrapper with a service interface including incoming messages and a request interface including outgoing messages as shown in Fig. 1.

In summary, roles specify not only what the system may request users to do, but also what users may ask for the system to do. They include two aspects: rights and responsibilities.

B. Role-Based Collaboration

By role-based collaboration, we mean that people collaborate in an environment where their roles are clearly specified. Its properties are as follows.

- **Clear role specification:** It is easy for human users to specify and understand their responsibilities and rights.
- **Flexible role transition:** It is flexible and easy for a human user to transfer from one role to another.
- **Flexible role facilitation:** It is easy for role facilitators to modify roles. Because collaborative activities are constantly evolving, even the existing roles might be required to adjust in correspondence with the development of the system.
- **Flexible role negotiation:** It is easy to negotiate a role’s specification between a human user and a role facilitator.
- The interactions among the collaborators are through roles.

From these properties, we find that roles are the key media for human users to interact and collaborate. The users are allowed to concentrate on interacting with other roles but not other human users to make the collaboration more operable. The role specification and negotiation are the major tasks to form an environment for collaboration. The specification of the interactions among roles actually defines the procedures of collaboration. Hence, role specification is the fundamental and key mechanism.
We propose the procedure of role-based collaboration as follows.

Step 1) **Negotiate roles.** People discuss or negotiate to specify the roles relevant to collaboration. If no agreement is reached, the collaboration aborts.

Step 2) **Assign roles.** Every person is assigned one or more role. If no agreement is reached, the collaboration aborts.

Step 3) **Play roles.** People work according to their roles until collaboration is successfully completed.

Step 3.1) **Check incoming messages.** People understand what they need to do at this time. The incoming messages are confined by the role responsibilities (the service interface). If there are conflicts or discontents, go to Step 1.

Step 3.2) **Issue outgoing messages.** People need to access and interact with the environment by sending messages, or asking for others’ services to provide their services. If there are no incoming messages, people may think and issue messages as they want. The messages are confined by their role’s rights (the request interface). If there are conflicts or discontents, go to Step 1.

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

Sometimes, we practice collaboration based on roles even though we do not declare so. Role-based collaboration is difficult to perform in natural environments because no role specification mechanism is available. Daily role negotiation cannot completely remove ambiguity. This is the major barrier to implementing role-based collaboration in the real world. Role-based systems can provide clarity and enhanced support in situations where collaborative working is required.

### IV. E-CARGO Model

Based on the role concepts discussed above and the basic principles for role-based collaboration [35], we can formalize the components of a role-based collaborative system. By establishing a formal model to define and specify a role, we can get a much clearer view of such systems. Without a clear specification of components, there would be no real successful systems. A well-defined internal structure is a guarantee for a successful system design and implementation. It supports the robustness, efficiency, and correctness of the entire system. The Appendix describes the internal structures and a formal model E-CARGO for role-based collaborative systems. By E-CARGO we mean Environments, Classes, Agents, Roles, Groups, and Objects. A system \( \Sigma \) can be described as a 9-tuple (see the Appendix for detail), \( \Sigma := (C, O, A, M, R, E, G, s_0, H) \), where

- \( C \) is a set of classes;
- \( O \) is a set of objects;
- \( A \) is a set of agents;
- \( M \) is a set of messages;
- \( R \) is a set of roles;
- \( E \) is a set of environments;
- \( G \) is a set of groups;
- \( s_0 \) is the initial state of a collaborative system;
- \( H \) is a set of users.

The initial state \( s_0 \) is expressed by initial values of all the components \( C, O, A, M, R, E, G, \) and \( H \). We include \( H \) to express that users might be affected by collaboration. Please note that without the participation of users, the system can only do what the agents can do. This is why users are an essential part of a system.

With a role specification mechanism, we can easily support role-based collaboration including both easy collaboration and flexible participations. We have to specify roles, build an environment, and let users play their roles in the environment for effective collaboration. In other words, when users want to work in a group, they should first play a role. They are then confined at this time by the role with special responsibilities and rights. When their current role is not satisfactory, they can ask for tuning their role or changing to another role. Without a role specification mechanism, we would have no way to do role transitions and tuning easily and efficiently.

In this model, we notice that a role has no knowledge about its environments and groups, because the environments and groups are derived components based on roles. As for the role of an individual, we should recognize that s/he can only be in one place at one time and is limited in the ability to move from place to place. In the above definitions, we note that roles are defined in the systems’ scope. Environments and groups are built after roles are specified. This truly reflects the reality of an organization.

The agent concept in our model E-CARGO has the potential property of autonomy. When a user X is logged out, the agent representing X will replace X to accept messages or send messages. When the agent class develops with more facts and knowledge, it would make better and better decisions in responding to incoming messages; it approaches a real agent.

The agents and roles are the media in which users operate the system and interact with others. Users can contribute to collaboration by doing the following:

- creating an agent class that mainly helps them automatically access objects, send outgoing messages and reply to incoming messages without their direct interference;
- negotiating roles with the role facilitator and negotiators;
- adding, deleting, and modifying the methods of an agent class and having the agent play roles;
- playing roles, i.e., accessing objects, sending outgoing messages, and replying to incoming messages.

To understand the significance of roles in a collaborative system, let us remove the role from the system \( \Sigma \). As a result,
$E$ and $G$ are also gone because they depend on roles. It now consists of only $C, O, A, M, s_0$, and $H$. $\sum$ becomes a traditional collaborative system. We lose the mechanism to organize collaboration and the media of interactions between the users and other components of the system. That is why traditional CSCW systems always argue about their multiple user interface design, because the interfaces are totally up to the implementations. Based on the E-CARGO model, the interfaces of a system are totally determined by the roles.

As for workflows, they can be formed by exchanging messages among roles. Suppose that we have roles $R_1$ with incoming messages $m_i11$ and $m_i12$ and outgoing messages $m_o11$ and $m_o12$; $R_2$ with $m_i21, m_i22, m_o21$, and $m_o22$; and $R_3$ with $m_i31, m_i32, m_o31$, and $m_o32$. If $m_o11$ matches $m_i22$, $m_o21$ matches $m_i31$, and $m_o21$ is required to respond to $m_i22$, we have a workflow $R_1 \cdot m_o11 \rightarrow R_2 \cdot m_i22 \rightarrow R_2 \cdot m_o21 \rightarrow R_3 \cdot m_i31$.

In our E-CARGO model, workflows can also be confined by the design of agent classes and their subclasses. Workflows are high-level “how-to-do,” and agent classes are used to specify some required common features in collaboration.

The scenario of a collaboration based on this kind of system can be described as follows.

- A collaborative system built with the role mechanisms is installed on a server.
- Each user uses an interface such as a web browser to sign up. A corresponding agent is created. Only a default role is assigned to the user.
- Users log into the system on client computers with default roles.
- Some users create classes, objects, agents, roles, environments, and groups in the system.
- Users negotiate roles and may play many roles but only one at a time. They may play different roles at different times based on the requirements of collaboration.
- A user can send messages with the request (right) interfaces relevant to their roles and get messages with the service (responsibilities) interfaces.
- The agents accept the incoming messages when the users log out.
- By timely negotiation, users can transfer roles and roles can be modified or tuned by other users with specific roles based on the requirements of collaboration.
- Through the roles, users access objects and their agents, contact other agents, join groups, and contribute to the collaboration.
- The result of the collaboration is reflected by the states of objects, roles, agents, environments, groups and users in the system.

In the system implementation, we avoid using natural languages to describe what a role should be. We use a group of messages or message patterns that have no or little ambiguity. By this strict definition, both role players and facilitators have a clear role specification without different role expectations so as to decrease role ambiguity. Certainly, there is still ambiguity (or positively flexibility) in how the role mechanism transmits the messages and how messages are processed by agents and human users.

In daily life, the roles of the members in a group may change. By these changes, different members may offer different contributions to the group even with the same role. That is to say, in reality, one member’s role is different from another’s role even if they have the same role name, such as assistant professor. The definition of roles seems to restrict the creative work of the members because a role is defined by certain message patterns. However, because message patterns are versatile, it is possible for message patterns to support the creative work of users. If the right to create objects is granted for a role, then the users who play this role can contribute all kinds of creative work. At the same time, by interactive negotiations among users, the roles’ responsibilities and rights can be modified to support the creative work of the users.

CSCW systems and intelligent collaborative systems are two different fields. The former supports people to collaborate with computer systems but the latter tries to design and implement intelligent computer systems in the same way as people’s collaboration. Collaborative intelligent systems hope to process incoming messages by agents themselves. However, in a CSCW system, people decide how to respond to a message.

In fact, these two kinds of systems can be combined. Our E-CARGO model is at first aiming at supporting people’s collaboration. With the development and the evolution of the systems built with E-CARGO and artificial intelligence (AI) technologies, a CSCW system might become more and more intelligent and could support more and more collaborative activities. This will become true after a role engine is built that automatically distributes messages to agents that are playing the role relevant to the messages and then reply to the requesting agents.

V. Kernel Mechanisms

Based on our discussion of role-based collaboration and the model E-CARGO, we can compose an architecture for building role-based collaborative systems. Object principles are one of the major groups of principles in role-based collaboration. Therefore, object-oriented technology is the first practical tool.

With an object-oriented paradigm, we can easily adopt a role’s incoming messages to express its services with class mechanisms. Therefore, we can use a special object to express a user providing specific services in a collaborative system. That is to say, an object provides specific services when playing roles.

Actually, users and their agents in a collaborative system as shown in Fig. 2 play the same role as persons in usual collaboration as shown in Fig. 1. Agents are treated to differ from ordinary objects.

RBAC has done much in dealing with users’ permission to certain operations but ignores the varieties of users’ incoming messages. We consider a role as two special interfaces, one called the request interface between a user and the system, and the other called the service interface between the agent representing a user and the system. In this view, roles are the real entities to allow users to communicate with the system.
In collaborative systems, when users log in to a system, they actually have two existences: their agents and themselves. Wherein, the agents take the position of the users when they log out. With the model E-CARGO, we can obtain a human–computer interaction scheme as shown in Fig. 2 and a structure of role-based collaborative systems shown as in Fig. 3. Please note that in Fig. 3 there are some agents without users linked to them. This means that the corresponding users are offline.

A. Client/Server Structure

To build a collaborative system as discussed above, we need to compose an appropriate architecture based on the client/server architecture correspondent to Fig. 3.

The clients support all the user interactions with the system:
- class and object management interfaces;
- role specification interfaces;
- role playing interfaces;
- role negotiation interfaces;
- 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 the users.

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

B. Primitive Roles

If users want to play a role their current agent class does not support, they must change their agent class or create another agent class, and then initiate it. If they want to change an agent class, they can change only the relevant one that has no subagent classes. Therefore, an agent should cover or support all the messages including incoming and outgoing messages of all the roles that users play. To facilitate role-based collaboration, we need to build the following primitive roles.

- Role facilitator: This role allows a user to specify roles. To specify a role, the user just needs to drag and push the required messages or message patterns from the lists of classes, agents, roles, groups, objects, and environments.
- Class manager: This role allows a user to create, delete, and modify classes. Newly changed classes are reloaded into the system.
- Object manager: This role allows a user to create, delete, and modify objects.
- Agent manager: This role allows a user to create, delete, and modify agents.
- Environment manager: This role allows a user to create, delete, and modify environments.
- Group manager: This role allows a user to create, delete, and modify groups.
- Role dispatcher: This role manages all messages sent to the roles managed by it. The situation is as follows. Users ask for services. They search and find one in their outgoing messages. They pick one and send it. When they are asked to specify a receiver among agents, objects, and roles, they select a role. This means that all the users who play this role are fine to serve them. A user who plays the role dispatcher will find a user/agent to respond to this message.
- Role negotiator: This role allows a user to approve another user who hopes to play and modify a role. The user who plays the role negotiator manages the role to be played by another user. If users want to play a role, they must provide their agent class that implements all the required incoming message patterns.

As for the primitive roles, we need to know what objects to manage. There must be an object that interacts with a user. For example, for the outgoing message “define a role” of the role facilitator, it is directed to the role class for its instantiation. This definition message is actually corresponding to a constructor of the role class.

To dispatch a message, when a role receives a message, it checks the tag of this message to differentiate all, any, or some messages, then it decides if the message is sent to all, any, or some agents. If it is an any-message, it evaluates the agents’ workloads and selects the agent with the lowest workload and sends this message to it.

As for role negotiation, when a user X logs in, X would see what roles have been assigned to X. If X does not want to play a role, X can reject it by a message to the role facilitator.

C. Fundamental Classes

To build a role-based collaborative system, we need to build the built-in classes such as object, class, agent, role, environment, and group. When users want to log into the system, they are presented a hierarchy of agent classes. They should select one (at least the class Agent) for them to build a new subagent class to instantiate. Therefore, agent classes should be organized in a system as a hierarchy shown in Fig. 4. This hierarchy tells us that a new agent is required to play specific roles.
If the selected agent class has only abstract methods, it means that no agent (or person) in the system has this ability. The users must implement or acclaim to accomplish this when they sign up. To specify the incoming message patterns, the role facilitator can select some of them from the agent classes. If a new responsibility is found, s/he should at first ask for or transfer to a class manager role and add one abstract method in an appreciate agent class and then add the message patterns to the new role.

D. Implementation

We have implemented the kernel components of E-CARGO with Java on the Eclipse integrated development environment (IDE) (www.eclipse.org). This implementation has demonstrated its feasibility. By “kernel” we mean that the work left to implement a role-based collaborative system is to implement the user-interface and the client/server communication. Java object references are used to implement identifications. The work also demonstrates that our role-based model supports the message interactions through roles. We assert that roles are the key media for collaboration. In our implementation, we modify the role structure shown in Fig. 1 to contain both messages and object references. This modification eases the implementation of this model. A C.Role class and a C.Agent class are developed. Every role is an instance of the class C.Role. Every agent is an instance of a subclass of the class C.Agent. Besides accommodating roles, the C.Role class provides mechanisms such as message dispatching with certain policies and role playing approval. The C.Agent class provides the mechanism for a user to play, replace, and delete a role. The roles and agents are coupled by reference links (Fig. 5).

To implement these two classes, the most important jobs are how to approve an agent to play a role and how to dispatch messages to agents. They are expressed by the methods addAgent() and dispatching() in C.Role.

The major ideas applied in the two functions are as follows.
- The agent to be added must have its class implement the methods corresponding to the required incoming messages of the role, i.e., the message patterns of the agent class must match all the message patterns of the incoming messages of the role.
- A message is dispatched to an agent by a role based on the message type such as “any,” “some,” and “all.” In our implementation, an “any” message is sent to the first available agent, a “some” message is sent to all the available agents, and an “all” message is sent to all the agents.

In our implementation, a group structure based on E-CARGO is shown in Fig. 5, where $A_i (i = 1, \ldots, m)$ expresses agents and $R_i (i = 1, \ldots, n)$ expresses roles. We can find that, for example, $A_1$ may play $R_1$ and $R_2$, and $A_2$ may play $R_2$. The object and class components can be implemented with the reflection mechanisms of the Java programming language.
Prototyping work has been limited to the server-side functionality. To build a whole system, we need client-side user interfaces and connections with the server. The client-side interfaces mainly present the information provided by the server.

VI. CONCLUSION

Based on the findings of social psychologists [1], [4], [15], [16], a role-based collaborative system allows users to know and fulfill their obligations while respecting the rights and authority of other users in collaboration. The following are the major contributions of this paper:

- establishing the requirements for role-based collaboration;
- proposing a generalized three-step process of role-based collaboration;
- describing a formal model E-CARGO that supports the fundamental principles of role-based collaboration and meets the requirements of role-based collaboration;
- proposing an architecture to implement role-based collaboration based on E-CARGO;
- implementing the kernel mechanisms of role-based collaboration, i.e., the key components of the E-CARGO model.

The major benefits of the formal model E-CARGO include the following.

- The role concept is supported. A role is independent of groups and can be taken as the central mechanism to support collaboration.
- A role is a manageable entity that can be flexibly modified and tuned.
- Agents can help users do some simple work when users are not online.
- A group is specified based on roles. This facilitates the group management tasks.

A comparison is shown in Table I, illustrating the significance of this work.

This paper has presented a formal basis to a real role-based collaborative system. There are many essential topics that require future research.

- Based on the presented kernel mechanisms, we need to build a practical role-based collaborative system. The algorithms for dispatching messages by the C_Role class need improvement. The mechanisms to support different types of messages (any, some, and all-messages) are needed. Such mechanisms are not trivial and require deep considerations including how to express the knowledge of agents and roles, how to dispatch messages, and how to avoid message conflicts.
- User-friendly client-side interfaces are needed for users to specify roles, and modify agent classes. The interfaces and the client/server components need to be developed to build a real distributed platform for role-based collaboration.
- Further research is needed to expand E-CARGO into a practical computer-aided design tool to develop role-based systems.
- Theoretical mechanisms and intelligent tools are demanded to facilitate role negotiations.

A role-based collaboration imposes challenges not found in traditional CSCW systems. The implementation of the kernel
mechanisms has demonstrated the feasibility of providing role-based models and kernel mechanisms.

APPENDIX
E-CARGO MODEL

This appendix describes a formal model E-CARGO for role-based collaborative systems. By E-CARGO, we mean Environments, Classes, Agents, Roles, Groups and Objects. Some basic terms are introduced first.

Identification is used to make differences between different entities in a system. Externally, it means an identifier composed of letters and digits. Internally, it means the initial address of a memory unit. Internal identification is unique but the external one is not. We use $N$ to express the set of all the identifications, $n$ a specific one, i.e., $n \in N$, and $nr, n_a, n_g$ are identifications of a role, agent, and group, respectively. We use identification to mean an internal one unless otherwise stated.

Data structure is a group of logical memory units used to accommodate data values. We use $S$ to express the set of all the data structures and $s$ a specific one, i.e., $s \in S$.

Human user is a person who has signed onto the system. It is shortened as “user” thereafter. We use $H$ to express all the users and $h$ a specific one, i.e., $h \in H$.

We use $a \cdot b$ to express $a$’s $b$ or $b$ of $a$.

A. Object and Class

From the viewpoint that everything in the world is an object and every object has a class, we could have the definitions for classes and objects.

Definition 1 (Class): $c ::= \langle n, D, F, X \rangle$, where
- $n$ is the identification of the class;
- $D$ is a data structure description for storing the state of an object including pairs of class and an external identification;
- $F$ is a set of the function definitions or implementations;
- $X$ is a unified interface of all the objects of this class. It is a set of all the message patterns relevant to the functions of this class. A message pattern tells how to send a message to invoke a function.

We use $c$ to express a specific class and $C$ the set of all classes.

Next, we define an object based on a class.

Definition 2 (Object): $o ::= \langle n, c, s \rangle$, where
- $n$ is the identification of the object;
- $c$ is the object’s class identified by the class identification or name;
- $s$ is a data structure whose values are called attributes, properties, or states.

We use $o$ to express a specific object and $O$ the set of all the objects.

B. Agent

An agent is defined as an entity consisting of a set of provided services [2]. An agent $a$ is a special object that represents a user involved in collaboration. It is created when a new user signs up to the system. The necessity of agents is that when a user does not log in, the agent could accept incoming messages and send out simple reply messages. Here, we emphasize that agents may not delegate all the outgoing messages of a role to let users contact the system directly. If agents could delegate all the outgoing messages of a role, the system would become an intelligent system.

Definition 3 (Agent): $a ::= \langle n, c_a, s, N_r, N_g \rangle$, where
- $c_a$ is a special class that describes the common properties of users;
- $n$ and $s$ have the same meanings as those in Definition 2;
- $N_r$ means a set of identifications of roles that the agent is playing;
- $N_g$ means a set of identifications of groups that the agent belongs to.

Agents are different from objects in that objects directly respond to messages by initiating their class methods, but agents...
may respond by presenting the messages to the user it represents or just transferring these messages to other agents. To support its functionality, an agent should have the knowledge about groups, classes, objects, roles, and other agents in the system. $A$ denotes the set of all the agents.

C. Message

Interaction is a necessary entity for collaboration. To facilitate interactions among roles, messages are used.

Definition 4 (Message): 
$$m := (n, v, l, P, t),$$

- $N$ 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;
- $P$ is a set of objects taken as parameters with the message pattern $l$, where $P \subset O$;
- $t$ is a tag that expresses any, some, or all messages.

A traditional object-oriented paradigm emphasizes the messages accepted by a class of objects. However, it does not consider much about the messages an object may send out. In this model, we emphasize that a role is a message receiver and a user sends messages to roles. In this definition, if $v$ is null, this message is an incoming one to be dispatched by the role to an agent and $P$ and $t$ are meaningless. If $v$ is an identification of a role, it is an outgoing message that should be dispatched by the role specified by the message. Outgoing messages should be filled with $P$ (it can be empty) and $t$. There are three categories of outgoing messages, i.e., any-message, some-message, and all-message. By any-message we mean that the message may be sent to any one agent who plays the role. Some-message means the messages should be sent to some agents and all-message means that the messages should be sent to all the agents who play the role.

We use $m$ to express a message and $M$ the set of all messages. We call $m$ a message template if its $P$ is not specified completely.

D. Role

A role can show the specialties of some users. It provides them with message patterns not only to serve others but also to access objects, classes, groups, and other roles.

Definition 5 (Role): 
$$r := (n, I, N_r, N_o),$$

- $n$ is the identification of the role;
- $I := (M_{in}, M_{out})$ denotes a set of messages, wherein, $M_{in}$ expresses the incoming messages to the relevant agents, $M_{out}$ expresses a set of outgoing messages or message templates to roles, i.e., $M_{in}, M_{out} \subset M$;
- $N_r$ is a set of identifications of objects including classes, environments, roles, and groups that can be accessed by the agents playing this role.

In a run-time system, $M_{in}$ is a subset of $X$ (interface) of $C_n$ (a set of special classes describing common properties of users) of an agent $a$ that plays this role. The elements of $M_{out}$ are constructed with the subsets of $M_{in}$ of other roles. Suppose that we have at least one agent playing a role, for roles $r_i$ and $r_j (i \neq j)$, where we have the equations as shown at the bottom of the page. $m_{in}$ expresses an incoming message $m_{in} \in M_{in}$ and $m_{out}$ an outgoing message ($m_{out} \in M_{out}$).

In this definition, accessing means obtaining all the services the objects provide. Note that we separate agents from objects to emphasize that a role is the media to access agents. The only way to interact with other agents is by sending messages to roles. Denote by $R$ the set of all the roles.

E. Environment

In reality, people collaborate in an environment. People normally build groups in an environment. We can mimic a play on a stage. The stage is the environment, the play or collaboration is performed by a group of actors. Therefore, we introduce a new concept to facilitate the definition of a group.

Definition 6 (Environment): 
$$c := (n, B),$$

- $n$ is the identification of the environment;
- $B$ is a set of tuples of role, number range, and an object set, $B = \{(n_r, q, N_r)\}$. The number range $q$ tells how many users may play this role in this environment and $q$ is expressed by [lower, upper]. For example, $q$ might be [1, 1], [2, 2], [1, 10], [3, 50], … It states that how many agents may play the same role $r_j$ in the group. The object set $N_r$ expresses the complex objects accessed by the agents who play the relevant role. By “complex” we mean they are composed of other objects. The complex objects in $N_r$ are categorized as two kinds: Singly owned or shared, i.e., one complex object in this set may be accessed by one agent or shared by many. In fact, $N_r$ expresses the resources for agents to access.

For example, a computer science department environment can be expressed in the following ways.

- $n = \text{Department of Computer Science}$.
- $B$ is as follows: One chairperson, one chairperson office; one to three associate chairpersons and one to three offices for associate chairpersons; one to five secretaries and one to five offices for secretaries; one to four computer system administrators and one to four offices for administrators; three to fifteen faculties and three to fifteen faculty offices expressed by \{\{chairperson, [1, 1], \{a chairperson office\}\}, \{associate chairperson, [1, 3], \{one to three offices for associate chairpersons\}\}, \{secretary, [1, 5], \{one to five offices for secretaries\}\}, \{computer administrator, ...\}.

\[
\forall m_{in}(m_{in} \in r_i \cdot M_{in} \rightarrow \exists a(m_{in} \in a \cdot G_a \cdot X \land a \cdot n \in r_i \cdot N_r))
\]

\[
\forall m_{out}(m_{out} \in r_i \cdot M_{out} \rightarrow \exists r_j(i \neq j, \exists m_{in}(m_{in} \in r_j \cdot M_{in} \land m_{in} \cdot l = m_{out} \cdot l)))
\]
[1, 4], {one to four offices for administrators}, {{faculty, [3, 5], {three to fifteen faculty offices}}},
Denote by \( E \) the set of all the environments in a system.

**F. Group**

Users work in a group and hold roles. Every work setting involves groups of individuals [36]. In a group, to accomplish a task, the group members (users) interact with each other. We can define a group as a set of agents representing users playing roles in an environment.

**Definition 7 (Group):** \( g = \langle n, e, J \rangle \), where
- \( n \) is the identification of the group;
- \( e \) is an environment for the group to work;
- \( J \) is a set of tuples of identifications of an agent and role, i.e., \( J = \{ (n_a, n_r, n_o) \mid \exists q_n, n_o \in N_o \land (n_r, q_n, n_o) \in e \cdot B \} \).

Suppose there are \( U = \{ (n_a, n_r, n_o) \mid \exists q_n, n_o \in N_o \land (n_r, q_n, n_o) \in J \} \) agents in the group, \( V = \{ (n_a, n_r, n_o) \mid (n_r, q_n, n_o) \in J \} \) roles in the environment \( e \) of a group and one agent plays exactly one role in the group. \( q_i \) means the role number range for the \( i \)th role. \( q_i \cdot \text{upper} \) means the upper limit of the number of the agents playing the \( i \)th role. \( q_i \cdot \text{lower} \) means the lower limit, we have the following inequality for a group

\[
\sum_{i=1}^{V} q_i \cdot \text{lower} \leq U \leq \sum_{i=1}^{V} q_i \cdot \text{upper}.
\]

It means that every agent must play a role in a group, an agent could join the group only when there are available vacancies for a role, and there must be a number of agents to play relevant roles. If we want the resources in an environment are used completely and without wastes, we should keep the equality as follows, for each \( b_i \in B \) in group \( g \), \( \sum_{i=1}^{V} b_i \cdot |N_o| \leq |J| \) means that there are more resources than required, and \( \sum_{i=1}^{V} b_i \cdot |N_o| < |J| \) means that there are not enough resources.

The Computer Science Department of Nipissing University is a group, where
- \( n = \) CS Department of Nipissing University;
- \( e = \) the Computer Science Department environment;
- \( J \) includes all the \( \langle \text{agent, role, object} \rangle \) tuples expressing its members, their roles, and their accessible offices.

The above definition of a group states this fact that without the users’ participation, no collaboration would be formed.

We use \( g \) to express a specific group and \( G \) the set of all groups. The relationship between an agent and a group is similar to that between an agent and a role.

After a group is built, users can log into the system, join it by playing a role, and access one relevant object in it, and interact with each other. To support a group to work is basically the routine function of a collaborative system. In this definition, for group \( g \), role \( r \), agent \( a \), and an object \( o \), we have such a relationship

\[ a \cdot n \in \{ b \cdot (b, r, o) \in g \cdot J \} \rightarrow a \cdot n \in r \cdot N_a. \]

This tells that joining a group means playing a role, but not vice versa, i.e., a role is independent of groups.

**G. Users**

To model a computer-based system, we cannot directly encompass people in it. Therefore, in this model, agents are representatives of people or users. All the formal and expressible properties such as profiles, simple automatic message processing, and well-accepted user models are expressed by agents. In a system, users show the inexpressible properties in collaboration such as natural intelligence and instant thinking.

**H. System**

**Definition 8 (System):** \( \Sigma ::= \langle C, O, A, M, R, E, G, s_0, H \rangle \), where
- \( C \) is a set of classes;
- \( O \) is a set of objects;
- \( A \) is a set of agents;
- \( M \) is a set of messages;
- \( R \) is a set of roles;
- \( E \) is a set of environments;
- \( G \) is a set of groups;
- \( s_0 \) is the initial state of a collaborative system;
- \( H \) is a set of users.

The initial state \( s_0 \) is expressed by initial values of all the components \( C, O, A, M, R, E, \) and \( G \), such as built-in classes, initial objects, initial agents, primitive roles, primitive messages, and primitive environments.

**References**


Dr. Zhu is a Member of Association for Computing Machinery and Life Member of the Chinese Association for Science and Technology, USA. He is the Cochair of the technical committee of Distributed Intelligent Systems of the IEEE SMC Society. He served as the Guest Editor for a special issue for the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS and the Guest Associate Editor for the International Journal of Pervasive Computing and Communications. He is the Program Committee Member for the 2003–2006 IEEE International Conference on SMC and five other international conferences. He was the recipient of the Best Paper Award from the 11th ISPE International Conference on Concurrent Engineering; the 2004 and 2005 IBM Eclipse Innovation Grant Awards; the Educator’s Fellowship of OOPSLA’03; a Second Class Nation-Level Award of Educational Achievement from the Ministry of Education of China (1997); a Second Class Nation-Level Award of Excellent Textbook from the Ministry of Education of China (2002); three First Class Ministry-Level Research Achievement Awards from DOD of China (1997, 1994, and 1991); and a Second Class Excellent Textbook Award of the Ministry of Electronics Industry of China (1996).

MengChu Zhu (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 the Rensselaer Polytechnic Institute, Troy, NY, in 1990.

In 1990, he joined the New Jersey Institute of Technology (NJIT), Newark, where he is currently a Professor of electrical and computer engineering and the Director of the Discrete-Event Systems Laboratory. He has over 200 publications including six books, over 80 journal papers, and 14 book chapters. He is the coauthor of Petri Net Synthesis for Discrete Event Control of Manufacturing Systems (Kluwer Academic, 1993); editor of Petri Nets in Flexible and Agile Automation (Kluwer Academic, 1995); coauthor of Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach (World Scientific, 1998); coeditor of Deadlock Resolution in Computer-Integrated Systems (Marcel Dekker, 2005); and coauthor of Object-Oriented Programming in C++: A Project-based Approach (Tsinghua University Press, 2005). He was invited to lecture in Australia, Canada, China, France, Germany, Hong Kong, Italy, Japan, Korea, Mexico, Taiwan, and U.S. His research interests include computer-integrated systems, Petri nets, wireless ad hoc and sensor networks, system security, semiconductor manufacturing, and embedded control.

Dr. Zhu is a Life Member of the Chinese Association for Science and Technology, USA, and has served as its President in 1999. He served as the Associate Editor of the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION and the Guest Editor for the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS and the IEEE TRANSACTIONS ON SEMICONDUCTOR MANUFACTURING. He is the Managing Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C, the Associate Editor of the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, and the Editor-in-Chief of the International Journal of Intelligent Control and Systems. He was the 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; the General Cochair of the 2003 IEEE International Conference on SMC; the Founding General Cochair of the 2004 IEEE International Conference on Networking, Sensors, and Control; and the General Chair of the 2006 IEEE International Conference on Networking, Sensors, and Control. He has organized and chaired over 70 technical sessions and served on program committees for many conferences. He was the Cochair (founding) of the Semiconductor Factory Automation Technical Committee of the IEEE Robotics and Automation Society. He has led or participated in 28 research and education projects with a total budget of over $10 million, funded by the National Science Foundation (NSF), Department of Defense, Engineering Foundation, New Jersey Science and Technology Commission, and industry. He was the recipient of the NSF’s Research Initiation Award, CIM University-LEAD Award by the Society of Manufacturing Engineers, Perkin 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, the Asian American Achievement Award by the Chinese American Heritage Council, New Jersey, and Distinguished Lecturer of the IEEE SMC Society.


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

From 1988 to 2000, he was a Lecturer, an Associate Professor, and a Full Professor at NUDT. From 1999 to 2002, he was a Visiting Professor and a Special Lecturer in the College of Computing Sciences, New Jersey Institute of Technology, Newark. Currently, he is an Assistant Professor at the Department of Computer Science and Mathematics, Nipissing University, North Bay, ON, Canada. He is the author or coauthor of more than 50 papers, four books, and one book chapter on object-oriented programming, distributed/collaborative systems, and computer architecture.