ABSTRACT

A society requires a government to manage, supervise and guide the activities of its members. Every system requires self control and supervision. A collaborative system also requires self management and control. Role-based systems have the same requirements.

In role-based collaboration (RBC), roles are the major media for interaction, coordination, and collaboration. A center is required to deal with the management of roles and role players, and the control of messages sent from roles to roles. This center is called as a role engine. In this paper, the basic functions of a role engine and the fundamental relations in RBC are discussed. These relations must be clearly specified before building a role engine and a role-based system.

KEYWORDS: roles, role engine, agents, relations, and role-based collaboration.

1. INTRODUCTION

Role-Based Collaboration (RBC) is a computational thinking methodology that mainly uses roles as underlying mechanisms to facilitate abstraction, classification, separation of concern, dynamics, interactions and collaboration. Based on roles, RBC is an emerging methodology used to facilitate an organizational structure, provide orderly system behavior, and consolidate system security entities that collaborate and coordinate their activities with or within systems.

An organization can be expressed in terms of a set of roles that determine the social position of agents within such an organization and their relations to other members of the organization. Agents should be designed in a way that enables them to reason about the social position they gain by adopting a specific role, and what the correct behavior in such a role would be [5].

Roles have three functions: specify special behavior (as interface roles), form the behavior of an agent (as process roles), and set a place for an agent in a group (or define the inter-relations among agents). Although many mention the relations among roles, there is no complete and comprehensive discussion on role relations.

Role specification includes two aspects: one is the content of a role and the other one is its relations with other roles. Role relation specification significantly affects almost all the aspects of role-based systems, such as, role modification, role assignment, role transfer, role execution and role interactions. The clear specification of role relations is fundamental to system analysis, system design and system construction.

This paper discusses the basic functions of a role engine and fundamental relations in RBC. Throughout, the terms agents and people are used interchangeably. If the collaboration is among system components, agents are considered to be autonomous components. If the collaboration is among people, agents are representatives of people in the collaboration. Our assumption is that an agent has only one processor (brain), can only perform sequentially, i.e., at one time, and can adopt only one role. This paper is arranged as follows. Section 2 depicts the basic functions of a role engine; Section 3 restates and revises our previously proposed model E-CARGO; Section 4 demonstrates the relations among roles; Section 5 defines the relations between roles and agents; Section 6 specifies the relations among agents; Section 7 presents the properties of an RBC system; Section 8 discusses related work; and Section 9 concludes this paper and proposes the future work.
2. THE FUNDAMENTAL FUNCTIONS OF A ROLE ENGINE

A role engine can be understood in the same way as a Prolog inference machine. For example, to use a Prolog system, people only need to write rules and facts. The Prolog inference machine will search the result. Similarly, to implement role-based collaboration, based on the proposed role engine, people simply need to specify roles and create agents based on role specifications. When agents are put into the role engine, the engine will drive agents’ work properly to obtain their goals by collaborating with other agents.

A role engine should do the following:
- Manage roles (create, delete, and modify);
- Manage agents (create, delete, and modify);
- Manage the credits (d) of agents;
- Assign roles to agents;
- Build role relations; and
- Check the consistency of the system.

A role engine is a platform for agents to collaborate. On this platform, agents work for the system by playing roles. Through implementation of the role engine, all agents are driven to contribute and work diligently for a system that offers a solution to a real-life problem. A role engine should possess role dynamics, facilitate role transfer, and support role assignments, interaction and presentation.

Multi-agent systems should encourage diversity of behavior in a population of cooperating agents [14]. There are definite reasons for the addition of new agents to the system. With well-built dynamics, agents will automatically follow the regulations of the RBC system and collaborate with each other to approach the common goal of the system. To build a role-based multi-agent system, the following questions should be answered: “How are agents created?”, “Why are agents created?”, “Why are agents transferred on the networks?”, “How are agents made pro-active?”, and “How could a system obtain the best performance?”. These questions concern the fundamental mechanisms to build and operate a system, i.e., dynamics.

The flexibility and survivability of a group (of people, systems, or system components) is largely dependent on its role assignments and role transferability [1, 3, 8, 17]. Although higher redundancy might imply a better system, it is important to seek out efficient solutions in allocating these resources. To solve such an optimization problem, the following questions should be answered clearly: What does “critical roles” mean? What does “critical people” mean? How are “critical roles and critical people” expressed? How is a crisis specified? Who is the best person to play a critical role? What are the criteria to evaluate if a person is qualified to play a role? How many roles should a person play or potentially play to deal with a crisis? Which roles should a person play? How can a crisis avoided by appropriate role transfer?

Successfully finding a new job is dependent on similarities between new roles and those previously performed by a person [2], i.e., qualifications are the basic requirements for possible role-related activities. To address role assignment problems, it is necessary to deal with issues arising from role definition and specification. Clear role definition and specification helps a person collaborate by avoiding ambiguities and conflicts [3]. Nobody likes to work in a group lacking clear regulations and rules. A highly efficient, productive society is well-organized, well-regulated and well-managed. Specifications are the key points. Initial questions such as “how is a role defined?” and “how are roles specified?” need to be answered.

Because the assignment of roles to agents is dynamic in a multi-agent system, i.e., the roles can be re-assigned to other agents based on the changing of the environment, such assignment needs to consider the following properties: Fairness: the engine should assign roles to agents based on a fair rule; Balanced workload: the engine should assign roles to qualified agents according to the workload of agents; and Least conflict: the engine should guarantee that the roles assigned to agents create the least opportunity for agents to have conflicts in sharing information when they are playing roles.

Dispatching messages to agents is a highly intelligent task to be accomplished by a role engine and its roles. It needs to consider several properties: Fairness: the engine should dispatch messages evenly to peer agents (see Definition 29); Consistency: the engine should check the consistency of role hierarchies (see Definition 38); and Completeness: sometimes, the engine should dispatch a specific message to all the agents of the targeted group.

Roles are finally presented to users and should be easily understandable. The specification of roles should consider the easy implementation of role presentation. Role presentation should give consideration to aesthetics, intuition, and other human factors.

3. E-CARGO MODEL AND REVISIONS

With the E-CARGO model [16], collaboration is based on roles. In E-CARGO, a system \( \sum \) can be described as a 9-tuple \( \sum := \langle C, O, A, M, R, E, G, s_0, \mathcal{H} \rangle \), 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, and \( H \) is a set of users. In such a system, \( A \) and \( H \), \( E \) and \( G \) are tightly-coupled sets. A human user and his/her agent play a role together. Every group should work in an environment. An environment regulates a group. With this tight coupling, it is emphasized that a role-based collaborative system is composed of both computers and human beings.

In this paper, roles and agents are concentrated on. Roles are taken as abstract interface definition as proposed in the E-CARGO model. The process meanings of roles are not applied. Agents are role players. The definitions of an agent and role are revised as follows, where, if \( \chi \) is a set, \( | \chi | \) is its cardinality: \( a.b \) means \( b \) of \( a \) or \( a \)’s \( b \).

**Definition 1:** role. A role is defined as \( r ::= \langle n, I, A_o, A_p, A_o, R_r, N_r \rangle \), where,
- \( n \) is the identification of the role;
- \( I ::= \langle M_{in}, M_{out} \rangle \) denotes a set of messages, where \( M_{in} \) expresses the incoming messages to the relevant agents, and \( M_{out} \) expresses a set of outgoing messages or message templates to roles, i.e., \( M_{in} M_{out} \subseteq M \);
- \( A \) is a set of agents who are currently playing this role;
- \( A_p \) is a set of agents who are potential to play this role;
- \( A_o \) is a set of agents who used to play this role;
- \( R_r \) is a set of roles interrelated with it (see Definition 16); and
- \( N_r \) is a set of objects that can be accessed by the agents playing this role.

**Definition 2:** agent. An agent is defined as \( a ::= \langle n, c_o, s, d, r, R_p, R_o, N_a \rangle \), where
- \( n \) is the identification of the agent;
- \( c_o \) is a special class that describes the common properties of users;
- \( s \) is the qualifications of the agent;
- \( r \) means a role that the agent is currently playing. If it is empty, then this agent is free;
- \( R_p \) means a set of roles that the agent is potentially to play \(( r, \notin a.R_p \) ); and
- \( R_o \) means a set of roles that the agent played before; and
- \( N_a \) means a set of groups that the agent belongs to.

All the current role and the potential roles of agent \( a \) (i.e., \( a.R_p \cup \{ a.r_i \} \)) form its repository role set, denoted as \( R_a \).

With the E-CARGO model, a role engine can be designed to support RBC in the way stated by Shakespeare “All the world is a stage \(( E, C, O \) ) , and all the men and women merely players \(( A \) ); they all have their exits and entrances \(( G \) ); and one man in his time plays many parts \(( R_0 \) ). (As You Like It, Act II, Scene 7)”. In the following discussions, \( r_0, r_1, r_2, ..., r_i, ..., r_j, ... \) are used to express elements in \( R \), and \( a, b, c, ..., (a \neq b, b \neq c, ...) \) in \( A \).

### 4. THE RELATIONS AMONG ROLES

The relations among people are complicated in social societies [6]. Role relations are more abstract than people’s relations. Based on the principle of information hiding in the abstraction process, role relations should be easier to master than people’s relations. To classify and clarify role relations helps build an efficient role engine. Role relations are foundations to implement the functions of a role engine.

#### 4.1. Role Classes and Instances

To specify role relations, it is required to specify what a role is. When roles are discussed, arguments about roles always occur. What are roles? Are roles classes? Are roles instances? What is a message to a role?

In role-based collaboration, we state that roles are instances of \( r \) in Definition 1 with properties of classes. \( r \) is actually a meta-class in the view of object-orientation. On the aspect of an instance, a role has its concrete agents and messages. On the aspect of a class, its cardinalities \(( |A_o|, |A_p|, |A_o| \) ) are actually to express how many instances are created for agents to play it. To classify this, we define role classes and role instances.

**Definition 3:** role class. Role classes are instances of \( r \).

**Definition 4:** role instance. A role instance is an instance of a role class.

In the definition of role of the revised E-CARGO model, role \( r \) is in fact a meta-class for all role classes. When a new role is defined, it is actually an instance of \( r \). When an agent plays a role, it plays a role instance linked with an instance of \( r \). In RBC and E-CARGO, there is no concrete role instance. Role instances are actually implemented by agents playing a role class. The number of role instances is restricted by the cardinalities of a role. In the following discussions, the relations at the level of role classes are concentrated on. In RBC, a message is at first sent to a role class. The role class dispatches the message to agents that are players of the role class.
4.2. Inheritance Relation

Roles themselves can be a classification tool. They are related in a classification hierarchy.

**Definition 5:** inheritance relation, super roles and sub roles. Role \( r_j \) is a super role of role \( r_i \) if \( r_i \) possesses all the properties of \( r_j \). Vise versa, \( r_i \) is a sub role of \( r_j \). An inheritance relation denoted as \( \Omega \) is a set of tuples of roles \(< r_i, r_j, \rangle \), where, \( r_j \) is a super role of \( r_i \) and \( r_i \) is a sub role of \( r_j \). Here, “inheritance” is similar to the inheritance concept of object-orientation.

Figure 1 shows the inheritance relations among a group of role classes.

**Definition 6:** root role and leaf role. Role \( r_i \) is called a root role if it has no super roles. Role \( r_i \) is called a leaf role if it has no sub roles.

In Figure 1, Professor is a super role of roles Full Professor, Associate Professor, Assistant Professor and Math Professor. Professor is a root role, and Math Full Professor, Math Associate Professor, ..., and CS Assistant Professor are leaf roles. In such a hierarchy, it is needed to clarify that an agent should be assigned with a leaf role, because leaf roles are concrete and executable by agents.

It is needed to check if this message is included in the role’s service set or in the roles’ sub role’s service set. If the message is in the sets and required agents are ready to respond to this message, the message sending is successful. In role-based systems, a role accepts and dispatches messages. A role cannot respond to messages directly.

4.3. Promotion Relations

In a well-designed community, there are pre-designed role promotion relations. This relation encourages the members in the community to work hard to contribute. It means that an agent could play an upper level role only when it has played the lower level role before.

**Definition 9:** role promotion, lower role and upper role. Role \( r_i \) is an upper role of role \( r_j \) if \( r_j \) must be played by agent \( a \) before \( r_i \) is assigned to \( a \). Vise versa, \( r_j \) is called a lower role of \( r_i \). A promotion relation denoted as \( \Lambda \) is a set of tuples of roles \(< r_i, r_j, \rangle \), where \( r_i \) is a lower role of \( r_j \) and \( r_j \) is an upper role of \( r_i \).

![Figure 2. Examples of Promotion Relations](image)

Role promotion is not natural but man-made. It is set in a role engine in order to be fair in role re-assignment of potential roles. It is set during the initialization and may be adjusted during collaboration.

Figure 2 gives two examples of promotion relations in the education and information technology fields.

4.4. Report-to Relations

A role provides many services. However, not all other roles can request its services and obtain instant responses. Some regulations are introduced to control the accessibilities of the services. For example, a team leader can tell a team member to complete a task but a team member cannot tell a team leader what to do [4]. This leads to a report-to relation.

**Definition 10:** report-to relation, supervisor role and supervisee role. Role \( r_i \) is a supervisor role of role \( r_j \) if
role \( r_i \) must respond to the requests from role \( r_j \) in a time limit set by \( r_j \). Vise versa, \( r_i \) is a supervisee role of \( r_j \). A report-to relation denoted as \( \Theta \) is a set of tuples of roles \(<r_i, r_j>\), where, \( r_i \) is a supervisee role of \( r_j \) and \( r_j \) is a supervisor role of \( r_i \).

From this definition, a rule can be designed in a role engine: an agent playing a supervisee role obtains punishments if it does not respond to its supervisor’s requests on time, i.e., the credit \( \delta \) of the agent is decreased. A report-to relation is different from a promotion relation in that an agent plays a supervisee role may be impossible to be promoted to its supervisor role and a lower role may not be supervised by its upper role.

4.5. Request Relations

Everybody serves others and everybody requests others’ services. Therefore, an evident relation between roles is the request role.

![Figure 3. A Request Relation](image)

**Definition 11:** request relation, service role and request role. Role \( r_i \) is called a request role of role \( r_j \) if \( r_j \) provides the services requested by \( r_i \), i.e., \( r_i.I.M_{\text{out}} \subseteq r_j.I.M_{\text{in}} \). Vise versa, \( r_j \) is called a service role of \( r_i \). A request relation denoted as \( \Theta \) is a set of tuples \(<r_i, r_j>\), where \( r_i \) is a request role of \( r_j \) and \( r_j \) is a service role of \( r_i \), i.e., \(<r_i, r_j> \in \Theta \) if \( r_i.I.M_{\text{out}} \subseteq r_j.I.M_{\text{in}} \).

The request relation also holds the irreflexive property. This property guarantees a role does not issue a request itself in order to save the offset of message passing. Note that, we exclude those roles requesting part of other roles’ services in the request relation. This is reasonable because we can split one such request role into several ones to follow this definition.

4.6. Derived Relations

The following relations are derived from the above basic relations. They are not independent.

**Definition 12:** competition relation and competitor role.

Role \( r_i \) is called a competitor role of role \( r_j \) if roles \( r_i \) and \( r_j \) have the same request role or the same upper role. A competition relation denoted as \( \Xi \) is a set of tuples of roles \(<r_i, r_j>\), where, \( r_i \) and \( r_j \) are competitor roles of each other. More exactly, \(<r_i, r_j> \in \Xi \) if \( \exists k \) \( \forall \( r_k, r_i \in \Theta \land \left( r_k, r_j \in \Theta \lor \left( r_k, r_i \in \Lambda \land \left( r_k, r_j \in \Lambda \right) \right) \right) \)

For examples, roles software consultant and software developers are competitor roles because they provide same services, i.e., software development and role client may request the same services from them; in Figure 2, roles researcher and assistant professor are competitor roles but roles programmer and researcher are not competitor roles. The competition relations are used to establish regulations for assigning roles and dispatching messages.

**Definition 13:** peer relation and peer role. Role \( r_i \) is a peer role of role \( r_j \) if \( r_i \) and \( r_j \) have the same supervisor role. A peer relation denoted as \( \Theta \) is a set of tuples of roles \(<r_i, r_j>\), where, \( r_i \) and \( r_j \) are peer roles of each other. More exactly, \(<r_i, r_j> \in \Theta \) if \( \exists k \) \( \forall \( r_k, r_i \in \Delta \land \left( r_k, r_j \in \Delta \right) \)

4.7. Conflict Relations

In role-based access control (RBAC), Nyanchama and Osborn point out that there are many conflicts when authorization of roles is concerned: user-user/group-user/group conflicts; role-role conflicts; privilege-privilege conflicts; user-role assignment conflicts; and role-privilege assignment conflicts. From the viewpoint of RBC, roles may be conflict when they are assigned to agents.

**Definition 14:** conflict relation, conflict roles. Roles \( r_i \) and \( r_j \) are conflict if one agent cannot play them together. \( r_i \) is called a conflict role of \( r_j \) and vise versa. A conflict relation denoted as \( \Xi \) is a set of tuples \(<r_i, r_j>\), where \( r_i \) and \( r_j \) are conflict roles of each other. More exactly, \(<r_i, r_j> \in \Xi \) if \( \exists k \) \( \forall \left( r_k, r_i \in A \land \left( r_k, r_j \in A \right) \right) \)

In a role-based system, the conflict relation is actually pre-defined in order to protect conflict roles from being assigned to the same agent.

4.8. Summary of Role Relations
As a summary, ℱ is used to express all the relations among roles in an RBC system. Suppose \( r_i, r_j \in R \) and \( r_i \neq r_j \), \( ℱ := \Omega, Λ, Δ, Θ, Ξ, ψ, ε, ζ \), where,

- \( Ω \): an inheritance relation. \( \prec_< r_i, r_j \in Ω \) if \( r_i \) inherits from \( r_j \).
- \( Λ \): a promotion relation. \( \prec_< r_i, r_j \in Λ \) if \( r_i \) is a lower role of \( r_j \) and \( r_j \) is a higher role of \( r_i \).
- \( Δ \): a report-to relations. \( \prec_< r_i, r_j \in Δ \) if \( r_i \) is a supervisor role of \( r_j \) and \( r_j \) is a supervisor role of \( r_i \).
- \( ψ \): a request relation. \( \prec_< r_i, r_j \in ψ \) if \( r_i, r_j \in Ψ \).
- \( ε \): a competition relation. \( \prec_< r_i, r_j \in ε \) if \( \exists \ r_k (\prec_< r_k, r_j \in Θ \land \prec_< r_i, r_k \in Θ) \lor (\prec_< r_k, r_i \in Λ \land \prec_< r_i, r_j \in Λ) \).
- \( θ \): a peer relation. \( \prec_< r_i, r_j \in θ \) if \( \exists \ r_h (\prec_< r_h, r_i \in Δ \land \prec_< r_i, r_h \in Δ) \).
- \( Ξ \): a conflict relation. \( \prec_< r_i, r_j \in Ξ \) if \( r_i \) and \( r_j \) are conflict.

The following properties should be kept in a role-based system.

**Property 1:** irreflexive. \( \forall r_i \in R (\prec_< r_i, r_i \not\in Ω) \land (\prec_< r_i, r_i \not\in Δ) \land (\prec_< r_i, r_i \not\in Θ) \land (\prec_< r_i, r_i \not\in Ξ) \land (\prec_< r_i, r_i \not\in ψ) \land (\prec_< r_i, r_i \not\in Ξ) \).

**Property 2:** transitive.
- \( \forall r_i, r_j, r_k \in R (\prec_< r_i, r_j \in Ω \land \prec_< r_j, r_k \in Ω \rightarrow \prec_< r_i, r_k \in Ω) \land (\prec_< r_i, r_j \in Λ \land \prec_< r_j, r_k \in Λ \rightarrow \prec_< r_i, r_k \in Λ) \land (\prec_< r_i, r_j \in Δ \land \prec_< r_j, r_k \in Δ \rightarrow \prec_< r_i, r_k \in Δ) \land \forall r_i, r_j, r_k \in R (\prec_< r_i, r_j \in Δ \land \prec_< r_j, r_k \in Δ \rightarrow \prec_< r_i, r_k \in Δ) \land \forall r_i, r_j, r_k \in R (\prec_< r_i, r_j \in ξ \land \prec_< r_j, r_k \in ξ \rightarrow \prec_< r_i, r_k \in ξ) \land \forall r_i, r_j, r_k \in R (\prec_< r_i, r_j \in ξ \land \prec_< r_j, r_k \in ξ \rightarrow \prec_< r_i, r_k \in ξ). \)

**Property 3:** symmetrical.
- \( \forall r_i, r_j \in R (\prec_< r_i, r_j \in Ω \rightarrow (\prec_< r_j, r_i \in Ω)) \land (\forall r_i, r_j \in R (\prec_< r_i, r_j \in ξ \rightarrow (\prec_< r_j, r_i \in ξ)) \land (\forall r_i, r_j \in R (\prec_< r_i, r_j \in Ξ \rightarrow (\prec_< r_j, r_i \in Ξ)). \)

**Property 4:** noncircular. There are not circles in inheritance, request, promotion, and report-to relations, i.e., \( \not\exists r_0, r_1, r_2, ..., r_n \in R, (\prec_< r_0, r_1, r_2, ..., r_n) \in X(X = Ω, Λ, Δ, Θ, Ξ, ψ, ε, ζ) \) and \( r_i \neq r_j \) for \( i \neq j \).

**Definition 15:** role graph. A role net is a directed graph \([13]\) formed by all the relations of \( ℱ \), denoted as \( ℱ,G \).

i.e., \( ℱ,G := < R, ℱ > \), where, \( R \) are the node set and \( ℱ \) are the edge set.

**Definition 16:** interrelated roles. Roles \( r_i \) and \( r_j \) are interrelated if \( \prec_< r_i, r_j \) belongs to the role graph, i.e., \( \prec_< r_i, r_j \in (Ω \cup Λ \cup Δ \cup ξ \cup ψ \cup Ξ \cup Ω \cup Δ \cup ε \cup Λ \cup Ξ \).

**5. THE RELATIONS BETWEEN ROLES AND AGENTS**

After roles and role relations have been set, a collaborative platform is built. Agents (actors) could collaborate (perform) on this platform (stage). Agents should be assigned with roles, behave on the platform, transfer roles and finally leave the platform, i.e., the relations between agents and roles should be established. Some relations are interrelated specified and some are established by executing predicates (operations).

**Definition 17:** current role/agent. Role \( r_i \) is the current role of agent \( a \) if \( a \) is currently playing \( r_i \), i.e., \( r_i = a \). At the same time, \( a \) is called as a current agent of role \( r_i \), i.e., \( a \in r_i.A \).

**Definition 18:** potential role/agent. Role \( r_i \) is a potential role of agent \( a \) if \( a \) is qualified to play but not currently playing this role, i.e., \( r_i \in a.R_a \). At the same time, \( a \) is called as a potential agent of \( r_i \), i.e., \( a \in r_i.A_a \).

From these definitions, one agent can only play one current role but have many potential roles, and one role may have many current and potential agents.

**Definition 19:** past role/agent. Role \( r_i \) is a past role of agent \( a \) if \( a \) used to play \( r_i \) but \( r_i \) is neither a current nor a potential role, i.e., \( r_i \in a.R_p \). At the same time, \( a \) is called as a past agent of \( r_i \), i.e., \( a \in r_i.A_p \).

**Definition 20:** apply-for \((a, r)\). The predicate is to have agent \( a \) apply for role \( r \).

After the predicate apply-for \((a, r)\) is received, the system should check if \( a \) is qualified to play \( r \) based on the qualifications of it.

**Definition 21:** approve\((a, r)\). It is a predicate that approves role \( r \) for agent \( a \). This event occurs when apply-for \((a, r)\) has been issued and the agent is evaluated to be qualified to play role \( r \). After this predicate is executed, \( r \) is added to the potential role set of agent \( a \), i.e., \( r \in a.R_p \).
Many agents may bid for one role. There must be a set of rules to select (approve) the best agent to play the role. It means that the approved agent should be the best one to accomplish the tasks specified by the role the highest efficiently and the least costly. Some system restrictions should be enforced when approving an agent: it can play a direct upper role of its repository roles, a leaf role, a lower role of its repository roles, a peer role and a past role but it cannot play a conflict role.

Definition 22: disapprove(a, r). It is a predicate that disapproves a role from an agent.

This predicate is executed in two situations: when an agent is promoted to play an upper role, this lower role is disapproved; when an agent is checked that its credit becomes below the required credit of the role, the agent is disapproved of this role. After it is executed, ri is deleted from the repository set and put into the past role set of agent a, i.e., ri ∉ a.Rp ∧ ri ∈ Rr.

Definition 23: transfer(a, r0, rj). It is a predicate that agent a transfers its current role from r0 to rj. After it is executed, rj is assigned to the current role of agent a and r0 is put back to the potential role set, i.e., (a.r = r0) ∧ (a.Rr = (a.Rr \ {r0}) \ {rj}). It can be executed only when the system is still workable role transfer occurs, i.e., all the roles have enough current agents.

Definition 24: dispatch(rj, a, m). It is a predicate that role rj dispatches message m to agent a. After it is executed, a does what m asks for.

Definition 25: reply(a, rj, p). It is a predicate that agent a replies an object p to role rj. After it is executed, rj completes one service and responds to its request role with object p.

6. THE RELATIONS BETWEEN AGENTS

Many agents are working together in collaboration. By playing different roles, they build different workflows and accomplish different tasks, information transmissions, and productions.

In collaboration, competition is inevitable. Therefore, agents may collaborate or compete. Agents may be collaborators or competitors.

Definition 26: collaborator. Agents a and b are collaborators if their roles are interrelated, ∃ ri, rj ∈ a.Rr ∩ b.Rr ∃ <ri, rj> ∈ Ω ∪ Λ ∪ Ω ∪ Ω ∪ ξ.

Definition 27: current collaborator. Agents a and agent b are current collaborators if they are currently playing interrelated roles or a.r and b.rr are interrelated, i.e., <a.r, b.rr> ∈ (Ω ∪ Λ ∪ Ω ∪ Ω ∪ ξ).

Definition 28: potential collaborator. Agents a and b are potential collaborators if they are not current collaborators but there are two interrelated roles in the intersection of their repository role sets, i.e., ∃ rj, ri ∈ a.Rr ∩ b.Rr ∃ <ri, rj> ∈ (Ω ∪ Λ ∪ Ω ∪ Ω ∪ ξ) ∧ <a.r, b.rr> ∈ (Ω ∪ Λ ∪ Ω ∪ Ω ∪ ξ).

Definition 29: peer. Agents a and b are peers if they play the same role, i.e., a.Rr ∩ b.Rr ≠ Ø.

Definition 30: current peer. Agents a and b are current peers if they have the same current role, i.e., a.r = b.rr.

Definition 31: potential peer. Agents a and b are potential peers if a.Rr ∩ b.Rr ≠ Ø ∧ a.r ≠ b.rr.

Definition 32: competitors. Agents a and b are competitors if their repository role sets contain competition roles, i.e., ∃ ri ∈ a.Rr ∃ rj ∈ b.Rr (a.rc ∩ b.rc ≠ Ø).

Definition 33: current competitor. Agents a and b are current competitors if their current roles are competitor roles, i.e., a.rc = b.rc.

Definition 34: potential competitor. Agents a and b are potential competitors if ∃ ri ∈ a.Rr ∃ rj ∈ b.Rr (a.rc ∩ b.rc ≠ Ø) ∧ a.Rr ∩ b.Rr ≠ Ø.

Definition 35: client and server. Agent a is called a client of agent b and b is called a server of a if ∃ ri ∈ a.Rr ∃ rj ∈ b.Rr (a.rc ∩ b.rc ≠ Ø).

Definition 36: current client and server. Agent a is called a current client of agent b and b is called a current server of a if a is currently playing a request role of the role b is currently playing, i.e., <a.r, b.rr> ∈ Θ.

Definition 37: potential client and server. Agent a is called a potential client of agent b and b is called a potential server of a if ∃ ri ∈ a.Rr ∃ rj ∈ b.Rr (a.rc ∩ b.rc ∩ Θ ≠ Ø) ∧ (a.r, b.rr) ∈ Θ.

7. PROPERTIES OF AN RBC SYSTEM AND THEIR APPLICATIONS
From the definitions 13, 26, 29 and 32, some interesting facts are presented (Figure 4): a) agents playing conflict roles are collaborators; b) peers may be collaborators or competitors; c) peer role players are collaborators; and d) clients and servers are collaborators.

For a), in sales, roles price setter and buyer of a specific item are conflict but two persons playing these roles are collaborators. In fact, two different agents are required to play different conflict roles collaborations. For b), in the review processes of scientific research, not all peers can be reviewers. To be fair, reviewers should be those peers who are neither collaborators nor competitors. For c), the people in an office collaborate to complete a task ordered by their common supervisor. For d), clients and servers collaborate to complete a service transaction in the service community.

Figure 4. Collaborators, Competitors, and Peers

Definition 38: **redundancy.** A system is redundant if some services are not requested, i.e., \( \exists (r_i \in R) \)

\( \forall (r_i.M_{\text{out}} \cup r_j.M_{\text{in}} (r_i \neq r_j)) \neq \Phi \), where, \( n = |R| \).

Definition 39: **insufficiency.** A system is insufficient if some requests are not served, i.e., \( \exists (r_i \in R) \)

\( \forall (r_i.M_{\text{out}} \cup r_j.M_{\text{in}} (r_i \neq r_j)) \neq \Phi \), where, \( n = |R| \).

Definition 40: **consistency.** A system is consistent if all the relations are consistent, i.e., it keeps the **Properties 1-4.**

The relations and the properties specified in this paper have been applied in a prototype of role-based chatting system [15]. The success of the prototype verified the discussed relations.

8. RELATED RESEARCH

Although roles as concepts have been discussed for many years, the literature lacks systematic descriptions of the relations among roles and their players. Most of them did not present role relationship specification and classification. References discussed here mention role relationships and activate the author’s work.

Marwell and Hage [6] present 100 role relationships in different societies including economic, political, health and welfare, science and education, family, religion, art and leisure sectors. Each role-relationship involves occupants, activities, locations and occurrences. All the relationships are classified in the senses of sociology, i.e., *gemeinschaft* and *gesellschaft*. Their analysis lacks abstraction in the sense of collaboration, management and systems and is not very useful in such fields.

In RBAC, Simon and Zurko [11] mention the role conflict relations when they discuss conflicts of interests relevant to complex tasks in a workflow management system. Conflicts common in business transactions to require two signatures before a check is issued and processes to avoid fraud. Nyanchama and Osborn [7] discuss the role-role conflict relations by role graphs. A role-role conflict means that the two roles should never appear together. This implies they should never be assigned to a single user, which is checked on user-role assignment. Their discussion concentrates on the conflict of permission authorizations to access system resources.

Skarmeas [12] proposes a role model to deal with the tree hierarchy of roles in organizations. Roles may contain other roles and/or elementary roles. In their proposed tree structures, elementary roles are leaves. In describing organizational relations, he introduces virtual roles to deal with the supervision relations among roles. Virtual roles are used to assign roles supervised by them to individuals. He mentions the role consistency problem mainly when role transfers occur. However, tasks of a role and roles themselves are not clearly separated.

Olarnsakul and Batanov [9] emphasize that roles have relationships (i.e., interactions and dependencies) with one another in order to fulfill assigned responsibilities. They present that roles and relationships are the building blocks of the organizational structure. In their model, a role relationship is a set of protocols that represent collaboration tasks or business processes.

Odell et al. [8] describe a metamodel for agents, roles, and groups which are incorporated with Unified Modelling Language (UML) notations. They believe that roles provide the building blocks for agent social systems. Roles meet the requirements of describing interactions among agents. They use associations to describe the relationships among roles. Classifier classes are introduced to illustrate the relationships among agents and roles. Their metamodel is a good reference to analyze and design systems with roles.

Ren et al. [10] present a coordination model, i.e., the Actor, Role and Coordinator (ARC) model. Roles are
taken as a key thrust in the ARC model and this is similar to the role dynamics mentioned in Section 2.1. In their model, behaviors of actors, roles and coordinators are formally defined and applied into supporting the reconfigurability and fault localization for open distributed embedded software systems. The specified behaviors regulate some relations in role-based systems, such as, memberships and configurations.

9. CONCLUSIONS

A role engine is a platform to support RBC. The basic functions of a role engine are clarified and the fundamental relations in RBC are formally defined. These definitions can be used to find more properties and analyze the design and construction of an RBC system. An RBC system is very complicated. To check and keep it consistent is a hard work. The formal definitions of relations would help check its consistency.

Building a workable role engine will require much additional research and practice. To fully accept RBC principles, one needs to accept that, in a society, it is the system structure formed by its roles that controls the people’s behavior. A role engine is just such a system needed to control agents’ behavior. To make a role engine reality, it is needed to answer the following questions: Are the relations completely presented? Should the role engine or roles collaborate to manage the dispatch of messages? What is the number relation among roles and agents? Should time and space factors be introduced into roles?

ACKNOWLEDGMENTS

This research is in part supported by National Scientific and Engineering Research Council, Canada (NSERC: 262075-06) and IBM Eclipse Innovation Grant. Thank Mike Brewes of Nipissing University for his proofreading this article. Thanks also go to the anonymous reviewers for their constructive comments.

REFERENCES