

Role-Based Systems are Autonomic

Haibin Zhu

Department of Computer Science and Mathematics, Nipissing University

100 College Drive, North Bay, Ontario, P1B 8L7, Canada

E-mail: haibinz@nipissingu.ca

Abstract

Autonomic Computing is a newly emerging computing paradigm in order to create computer systems capable of self-management and overcome the rapidly growing complexity of computing systems management.

To possess self-* properties, there must be mechanisms to support self-awareness, i.e., an autonomic system should be able to perceive the abnormality of its components. After abnormality is checked, a process of self-healing or self-configuration must be completed to guarantee the system works correctly and continuously.

In role-based collaboration (RBC), roles are the major media for interaction, coordination, and collaboration. A role can be used to check if a player behaves well or not. This paper investigates the possibility of using roles and a role engine to diagnose the behavior of agents, and facilitate self-* properties of a system. The paper asserts that role-based systems are autonomic.

Keywords: roles, agents, role-based systems, and autonomic computing.

1. INTRODUCTION

The autonomic computing paradigm was proposed to deal with the increasing complexity in computer-based systems, and the shortage of skilled human manpower to advance the related technological developments and innovations. Autonomic computing is to simulate the natural neural system of human bodies to possess autonomic properties, i.e., self-* properties [8, 10, 13]: **self-awareness:** an autonomic system “knows itself” and is aware of its state and its behaviors; **self-configuring:** an autonomic application should be able to configure and reconfigure itself under varying and unpredictable conditions; **self-optimizing:** an autonomic system should be able to detect suboptimal behaviors and optimize itself to improve its execution; **self-healing:** an autonomic system should be able to detect and recover from potential problems and continue to function smoothly; and **self protection:** an autonomic system should be capable of detecting and protecting its resources from both internal

and external attack and maintaining overall system security and integrity.

Three basic ways are used to make a system autonomic [11]: to build an isomorphic space of possible system configurations; to design planned capabilities and social skills to delegate tasks to components; evolutionary approaches such as making components as biological cells. Role-based approaches belong to the second category.

In fact, a human body is composed of numerous cells and cell groups. All the cells and cell groups are collaborating and accomplishing different tasks to provide the functions of its host - a human body [21]. Because a human body is in a micro-world, it is easier to understand the autonomic properties of a system with a society in a macro-world.

A person is social. Without a society, a person never shows and requires his or her sociability. Without the requirement of sociability, one does not need to present their autonomy and autonomicity, because they coexist with the sociability. In a society, people work, interact, and collaborate with each other directly and roles are abstract regulations for them to collaborate and coordinate. In such situations, roles are abstract and implicit tools to evaluate people’s performance in completing tasks. This kind of evaluation is ambiguous and qualitative most of the time. Therefore, it is difficult to operate role-based evaluations in the social world. However, roles can be built into easily managed mechanisms to evaluate agents’ performances in computer-based systems [24, 25, 26].

In a multi-agent system, there are two kinds of “selves”: one is the system itself and the other is an agent itself. To efficiently present the autonomic properties of the system self, we must fully develop the autonomic properties of the agent self. To present the system self, there must be task distributions, coordination and collaboration among agents. To make agents autonomic, they must be aware of their environments, respond to these environments, and be self-regulating.

The self-management of a society comes from the good central government and the active participation of their members. The self management of a networked computer system also requires the central management

and the participations of their component computers. From the viewpoint of multi-agent system, an autonomic system is composed of a central manager and many autonomous and autonomic agents.

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 such an emerging methodology designed to facilitate an organizational structure, provide orderly system behavior, and consolidate system security for both human and non-human entities that collaborate and coordinate their activities with or within systems.

This paper is arranged as follows. Section 2 restates and revises the definitions of roles and agents of our previously proposed model E-CARGO; Section 3 investigates autonomic elements in role-based systems; Section 4 demonstrates how roles help a system possess self-awareness; Section 5 discusses the self-healing and self-configuration properties of a role-based system; Section 6 discusses self-protection; Section 7 presents the self-optimization property; Section 8 discusses related work; and Section 9 concludes this paper and proposes future work.

2. A REVISED E-CARGO MODEL AND ROLE-BASED SYSTEMS

With the E-CARGO model [26], collaboration is based on roles. In E-CARGO, a system Σ can be described as a 9-tuple $\Sigma ::= \langle C, O, \mathcal{A}, \mathcal{M}, \mathcal{R}, \mathcal{E}, \mathcal{G}, s_0, \mathcal{H} \rangle$, where C is a set of classes, O is a set of objects, \mathcal{A} is a set of agents, \mathcal{M} is a set of messages, \mathcal{R} is a set of roles, \mathcal{E} is a set of environments, \mathcal{G} is a set of groups, s_0 is the initial state of a collaborative system, and \mathcal{H} is a set of users. In such a system, \mathcal{A} and \mathcal{H} , \mathcal{E} and \mathcal{G} are tightly-coupled sets. In role-based collaboration, 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, a role-based collaborative system is composed of both computers and human beings. With the participation of people \mathcal{H} , such as joining in a team Σ , accessing objects of the team, sending messages through roles, forming a group in an environment, Σ evolves, develops and functions. The results of the team work are a new state of Σ that is expressed by the values of $C, O, \mathcal{A}, \mathcal{M}, \mathcal{E}, \mathcal{G}$, and \mathcal{H} . To apply role-based collaboration methodologies into autonomic computing, agents \mathcal{A} will be incorporated with some functions of human beings \mathcal{H} .

In this paper, roles, agents and environments are major mechanisms to support autonomic properties. Roles are considered abstract interface definitions 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 χ is a set, $|\chi|$ is its cardinality; $a.b$ means b of a or a 's b ; " $a=b$ " means a is equivalent to b ; " $a:=b$ " means that a is assigned with b ; and " $a\equiv b$ " means that a is defined with b . In E-CARGO, agents are considered to have only one central processing unit and they can only play one role and process related messages at one time.

Definition 1: *role*. A role is defined as $r ::= \langle n, I, s, d, \alpha, \beta, \mathcal{A}_c, \mathcal{A}_p, \mathcal{A}_o, \mathcal{R}_i, O_r \rangle$ where,

- n is the identification of the role;
- $I ::= \langle \mathcal{M}_{in}, \mathcal{M}_{out} \rangle$ denotes a set of messages, where \mathcal{M}_{in} expresses the incoming messages to the relevant agents, and \mathcal{M}_{out} expresses a set of outgoing messages or message templates to roles, i.e., $\mathcal{M}_{in}, \mathcal{M}_{out} \subset \mathcal{M}$;
- s is the qualification requirement for an agent to play this role;
- d is the credit requirement for an agent to play this role;
- α is the space limit for an agent to play this role; β is the time limit for an agent to play this role.
- \mathcal{A}_c is a set of agents who are currently playing this role;
- \mathcal{A}_p is a set of agents who are qualified to play this role;
- \mathcal{A}_o is a set of agents who used to play this role;
- \mathcal{R}_i is a set of roles interrelated with this role [24]; and
- O_r is a set of objects that can be accessed by the agents playing this role;

Definition 2: *agent*. An agent is defined as $a ::= \langle n, c_a, s_a, \alpha_a, \beta_a, r_a, \mathcal{R}_p, \mathcal{R}_o, \mathcal{N}_g \rangle$, where

- n is the identification of the agent;
- c_a is a special class that describes the common properties of users;
- s_a is the qualifications of the agent;
- α_a is the time requirement of the agent;
- β_a is the memory space requirement of the agent;
- r_a means a role that the agent is currently playing. If it is empty, then this agent is free;
- \mathcal{R}_p means a set of roles that the agent is potentially to play ($r_c \notin a.\mathcal{R}_p$);
- \mathcal{R}_o means a set of roles that the agent played before; and
- \mathcal{N}_g means a set of groups that the agent belongs to.

All the current role and the potential roles of agent a (i.e., $a.\mathcal{R}_p \cup \{a.r_c\}$) form its repository role set, denoted as \mathcal{R}_a .

Definition 3: *message*. $m ::= \langle n, v, l, p, f, \alpha_m, \beta_m \rangle$ where

- n is the identification of the message;
- v is 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 complex object taken as the parameters with the message pattern l ;
- f is a flag that expresses any, some or all-message;
- α_m is the space limit for this message to be processed; and
- β_m is the time limit for this message to be processed.

Definition 4: *environment*. $e ::= \langle n, \mathcal{B}, O_e \rangle$ where

- n is the identification of the environment; and
- $\mathcal{B} = \{ \langle r, q, o_r \rangle \}$ is a set of tuples of role, role range and a shared object. The cardinality range q tells how many agents may play this role in this environment and is expressed by $[l, u]$.

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 in the group. The object o_r expresses the object accessed by all the agents that play the role r .

Definition 5: $g = \langle n, e, \mathcal{J} \rangle$ where

- n is the identification of the group;
- e is an environment for the group to work; and
- \mathcal{J} is a set of tuples of identifications of an agent and role, i.e., $\mathcal{J} = \{ \langle a, r, n_o \rangle \mid \exists q \in Q, o_r \in O \exists \langle r, q, o_r \rangle \in e.\mathcal{B} \}$.

$\mathcal{E}, \mathcal{C}, \mathcal{A}, \mathcal{R}, \mathcal{G}, \mathcal{O}$ and \mathcal{M} are used to express the whole set of environments, classes, agents, roles, groups, objects and messages. Q is used to express the whole set of pairs of positive integers, such as, $[l, u]$. 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 ($\mathcal{E}, \mathcal{C}, \mathcal{O}$), and all the men and women merely players (\mathcal{A}); they all have their exits and entrances (\mathcal{G}); and one man in his time plays many parts (\mathcal{R}). (As You Like It, Act II, Scene 7)”. Role-based approaches are good candidates to build autonomic computing systems, because “autonomic computing technology must work in real world scenarios [12].”

Definition 6: *workable role*. A role r is workable in an environment e if it is assigned enough agents to currently play it, i.e., $\exists q \in Q, o_r \in O \exists \langle r, q, o_r \rangle \in e.\mathcal{B} \wedge (|\mathcal{A}_c| \geq q.l)$. This is denoted as $workable(r) = T$.

Definition 7: *workable group*. A group g is workable if all its roles are workable, i.e., $\forall r \in \mathcal{R} (\exists q \in Q, o_r \in O \exists \langle r, q, o_r \rangle \in g.e.\mathcal{B} \rightarrow (workable(r) = T))$.

With E-CARGO, a role-based system is composed of *groups* of *agents* playing *roles* situated in an *environment* to access *classes* of *objects* (Fig. 1).

With role-based approaches, a role engine is implemented on an operating system. The role engine manages roles and agents. From this view point, a process is composed of an agent and its current role. When a process is scheduled, an agent and its current role (r_c) and potential roles (\mathcal{A}_p) are loaded into the memory space of this process and then the code segment of the agent is executed on the CPU.

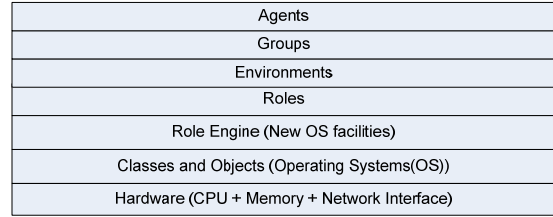


Fig. 1 A role-based system

In this view, context switch has two levels: one is agent switch and the other is role switch. Agents are scheduled first, followed by roles. Agent switch is similar to the heavy-weight process switch and role switch is similar to the light-weight process (thread) switch.

Roles are different from threads. Roles are interfaces and objects accessed by the agents. A role switch makes an agent execute different code segments originally stored in the memory space of an agent. A role is mainly used to regulate the behaviors and check the performance of an agent. The accessibility of an agent is restricted by its current role.

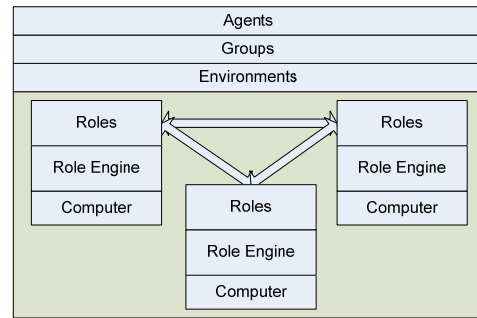


Fig. 2 A role-based distributed system

In a role-based distributed computer system (Fig. 2), roles are designed and stored in different computers including hardware and OS. Roles are the medium for inter-computer communications through their message exchanges. An environment can be organized with roles without considering their locations. This establishes distributed groups.

Agents are designed and stored in the peripheral memories of different computers. Agents are assigned both current and potential roles. Running agents are loaded into the main memory and scheduled for

processing on the CPUs. Agents transfer their roles based on the commands of role engine. After agents are assigned roles located on various computers in an environment, a distributed group is established. Group workability is dependent upon role workability.

3. AUTONOMIC ELEMENTS

An autonomic element manages its own internal state and interactions with its environment (i.e., other autonomic elements) [13]. It consists of a managed component and an autonomic manager [15, 18, 19]. An element's internal behavior and its relationships with other elements are driven by the goals and policies the designers have built into the system. Control loops with sensors and effectors together with system knowledge and planning/adapting policies allow the autonomic elements to be self-aware and self-management.

An autonomic element consists of one autonomic manager and one or more managed elements. At the core of an autonomic element is a control loop that integrates the manager with the managed element. The autonomic manager consists of sensors, effectors, and a five-component engine including a monitor, an analysis engine, a planning engine, an execution engine and a knowledge base.

An autonomic manager must be armed with tools for monitoring the elements managed by it [11]. It needs to analyze the collected data to determine whether the elements behave as expected, to plan a series of actions if a problem is detected, and to tune the parameters of the elements. With role-based approaches, the role engine (denoted as \mathcal{R}) and roles (Fig. 2) take the role of the autonomic manager and agents are managed elements. The regular tasks of a role engine include binding agents and roles [24]:

- $credit(a)$: return the credit of agent a .
- $set-credit(a, d)$: set the credit of agent a as d
- $qualified-for(a, r)$: return T if agent a is qualified to play role r else F.
- $apply-for(a, r)$: deal with an application for role r from agent a .
- $approve(a, r)$: check if agent a is appropriate to play role r .
- $disapprove(a, r)$: deprive agent a 's right to play role r .
- $misbehaving(a)$: check agent a 's current working state. It returns T if a does not behave well else F.
- $transfer(r, a)$: transfer role r 's current agent a to another qualified one. It returns T if successful else F.
- $address(m)$: return the source address of message m . Note that the identification of a message includes

the source address, such as, the port number and the Internet Protocol (IP) address.

- $dispatch(r, a, m, t_d)$: dispatch message m received by role r to agent a at t_d . It returns T if successful else F.
- $reply(a, r, m, p, t_r, c)$: return the result object p produced by agent a to role r with respect to message m at t_r with the space c accessed by a . It returns T if successful else F.

The role engine specifies the regulations, rules and policies of role playing [24], such as, role transfer and role promotion. The role engine is to maintain and keep the system sufficient, necessary and consistent:

- every request has a service;
- every service has a request;
- every role is workable; and
- there is no circulation among the role relations such as inheritance, request, report-to, and promotion relations.

Roles are added to a role engine based on the requirement of an environment of a system. When a role is added, there are vacant positions (the cardinality range of the role) for agents.

With E-CARGO, an agent is an instance of the agent class *Agent* defined in the model. Agents are created and added to a role engine based on the requirements of newly added roles [22]. Agents are made simpler because the role engine takes many controlling responsibilities from agents. The goal for an agent is to collect as many credits as possible. It needs to work hard: try to play roles, try to serve as many messages as possible, and try to play upper roles. An agent checks the vacant positions from an environment, checks if it is qualified to play the roles with vacant positions; then bids for the positions with other agents.

Definition 8: *qualified agent.* Agent a is qualified to play role r if a 's qualifications cover the qualification requirement of r , i.e., $\forall r \in \mathcal{R}, a \in \mathcal{A} (r.s \subseteq a.s_a) \rightarrow (qualified-for(a, r) = T)$.

Agents are required to adjust their time and memory parameters α_a and β_a timely to reflect its current performances. Suppose there is a method defined in a subclass of a corresponding to message m . Agent a is an instance of a subclass of the following Java-like class specification *Agent*, its major loop is shown in $run()$, where, *thread* is a Java class, and $sleep()$ is a method of class *thread* used to set a thread to an inactive state.

```
abstract class Agent extends thread {
    void run(){
        while (true){
            while ( $\exists e \in \mathcal{E}, r \in \mathcal{R}, q \in \mathcal{Q}, o_r \in \mathcal{O} \exists \langle r, q, o \rangle \in e.B \wedge (r.A_c \leq q.u) \wedge (\mathcal{R}.qualified-for(a, r) = T)$ )  $a.sleep()$ ;
```

```

ℳ.apply-for (a, r);
while (!ℳ.approve (a, r)) a.sleep();
while (!ℳ.dispatch(r, a, m, td) a.sleep();
ℳ.reply(a, r, m, a.m(), tr, c);
}
}
}

```

4. SELF AWARENESS

A person's body possesses the ability to self diagnose. People will wish to see a doctor when they do not feel well. There is then a need to express the nature of the malady. Therefore, a feeling of some abnormality is the first step in seeking treatment by a doctor. If a person cannot sense evidence of an illness, there would be no obvious need to obtain a doctors' assistance in seeking a cure for his/her medical problem. Autonomic systems have similar requirements. Self-diagnosis is the first step for autonomic computing to become a reality. "Problem localization and/or determination, the challenges become especially interesting [10]." How is agent behavior to be evaluated? A typical traditional way is to probe failures, i.e., to issue a ping command to a component. If the responses are out of the time limit, it can be taken as a possible indication of a bad component [10].

Roles provide new mechanisms to meet this requirement. In human societies, people (detectives and police persons) detect the abnormal behavior of some possible criminals by comparing their behaviors with the required behaviors by their declared roles. In computer-based systems, roles can also perform similar tasks. Roles are templates to regulate software agents' behavior and the interactions among agents. In a multi-agent system, agents are composed of qualifications and abilities. They behave based on their pre-programmed processes. An agent behaves abnormally when its pre-programmed processes are infected by malicious attacks. These kinds of agents are called **infected agents**.

The basic assumption is that an infected agent cannot complete routine tasks in the required time and within a preset space specified in its current role. It must take extra time and extra space to do some other jobs that are unrelated with its required tasks. For example, if a criminal wants to do some bad things, s/he must pretend to do the regular routine tasks. S/he must take more time than regular time consumption and perform activities in more places than the regular ones. Criminals may lie to find some excuses in social life to avoid penalty. However, such lies can be easily checked in computer-based systems because the CPU cycles used by an agent are definite, exactly measured and recorded.

When an infected agent is running, it actually takes extra CPU time and memory space to behave according to as its affected code. The role engine has a time slot to

run to check every agent's performances and report if an agent is ill.

A role's main content is composed of messages. There is no code in a role. Therefore, it is not easy for a role to be affected by malicious intruders. The role engine can compare the parameters specified in the role and the actual performance parameters of the agent to report if it is affected.

Rules (1-4) show the conditions for finding a misbehaving agent, i.e., (1) if the message is not correctly processed and replied; (2) if the message is processed in a too long period; (3) if the message is processed with a too large space; (4) if the collect space or time is over the limit; then the agent misbehaves. Rule (5) is to set the credit of the misbehaving agent as zero. It is reasonable to set the fourth rule to check if an agent is infected because each message process may not be significantly enough to express the agent's wrong behavior. At the role level, it is easier to check the agent's state by accumulating its tiny bad behaviors.

$$\forall a \in \mathcal{A} (\exists m \in \mathcal{M}, \exists r \in \mathcal{R} (\text{dispatch}(r, a, m, t_d)=T) \wedge (\text{reply}(a, r, m, p, t_r, c)=F)) \rightarrow (\text{misbehaving}(a)=T) \quad \text{---- (1)}$$

$$\forall a \in \mathcal{A} (\exists m \in \mathcal{M}, \exists r \in \mathcal{R} (\text{dispatch}(r, a, m, t_d)=T) \wedge (\text{reply}(a, r, m, p, t_r, c)=T) \wedge (t_r - t_d > m.\alpha_m)) \rightarrow (\text{misbehaving}(a)=T) \quad \text{---- (2)}$$

$$\forall a \in \mathcal{A} (\exists m \in \mathcal{M}, \exists r \in \mathcal{R} (\text{reply}(a, r, m, p, t_r, c)=T) \wedge (c > m.\beta_m)) \rightarrow (\text{misbehaving}(a)=T) \quad \text{---- (3)}$$

$$\forall a \in \mathcal{A} (a.\alpha_a \leq a.r_c.\alpha) \vee (a.\beta_a \leq a.r_c.\beta) \rightarrow (\text{misbehaving}(a)=T) \quad \text{---- (4)}$$

$$\forall a \in \mathcal{A} (\text{misbehaving}(a)=T) \rightarrow (\mathcal{I}.set_credit(a, 0)) \quad \text{---- (5)}$$

There may be an argument that fake infect reports are possible for rules (3) and (4) because a message m may have a large parameter p . This can be overcome by special design considerations, i.e., the spaces used for processing the message and storing the parameter p are separate and checked with different standards.

5. SELF HEALING AND SELF CONFIGURATION

When abnormality is found, it should be reported or recorded by a manager of the system. The manager is called role engine in a role-based system. The role engine will respond to the report and take action to cure the abnormality. The misbehaving agent should be stopped and replaced in a limited time. The major curing method is to transfer the role currently played by the infected agent to another qualified and uninfected agent. In fact, the situation is to replace the current abnormal agent by a normal qualified agent.

With E-CARGO, qualified agents are actually those agents belonging to its repository role set, i.e., \mathcal{A}_r , \mathcal{A}_r and related algorithms reflect the ability of a system to recover. A benefit of role-based systems is that once an agent is evaluated to be an infected one, it will never be assigned opportunities to run until it is cleaned or cured, because it cannot pass the qualification checking of the role. Rule (6) invokes a role transfer, i.e., the current role of agent a will be currently played by agent a' and the current role of a is deprived.

$$\forall a \in \mathcal{A} ((a.d = 0) \rightarrow \text{transfer}(a.r_o, a) \wedge \text{disapprove}(a, a.r_c)) \quad \text{---- (6)}$$

To have the ability of self-configuration [4, 15], finding qualified agents instantly are required. This is when there are not enough role pre-assignments, i.e., there exists not a successful role transfer [23, 27].

To meet this requirement, the role engine should actively search from all the available agents. Therefore, a new operation should be added to the role engine:

- *search-for-instantly*(a, r): find an qualified agent a for role r in the agent set and assign it to role r , i.e., $a.r_c = r \wedge a \in r.\mathcal{A}_c$.
- *search-for* (a, r): find a qualified agent a for role r in the agent set and put it into the set of qualified agents of role r , i.e., $(a.\mathcal{R}_p := a.\mathcal{R}_p \cup \{r\}) \wedge (r.\mathcal{A}_p := r.\mathcal{A}_p \cup \{a\})$.

For instant self-configuration, rule (7) is applied.

$$\forall a \in \mathcal{A} ((\text{transfer}(a.r_o, a) = F \rightarrow \text{search-for-instantly}(a', a.r_c)) \quad \text{---- (7)}$$

For a system expressed by g to re-configure before an instant searching is required, rule (8) is applied.

$$\forall r \in \mathcal{R} ((\exists q \in \mathcal{Q}, o_r \in \mathcal{O} \exists \langle r, q, o_r \rangle \in g.e.\mathcal{B}) \wedge (|r.\mathcal{A}_c| < q.u) \rightarrow \text{search-for}(a, r)) \quad \text{---- (8)}$$

6. SELF PROTECTION

In Role-Based Access Control (RBAC), roles possess the following characteristics which are beneficial to self-protection [2, 3, 6, 7, 9, 14, 17]:

- Least Privilege: It requires that outsider agents be given no more privileges than necessary to perform their job function.
- Separation of concerns: (1) a role can be associated with an operation of a business function only if the role is an authorized role for the subject and the role was not be assigned previously to all of the other operations; (2) an outside agent is authorized as a member of a role only if that role is not mutually exclusive with any of the other roles for which the agent already possesses membership; and (3) a subject can become active in a new role only if the proposed role is not mutually exclusive

with any of the roles in which the subject is currently active.

- Cardinality: The capacity of a role cannot be exceeded by an additional role member.
- Dependency constraints: There is a hierarchy or relationships among roles such as *contains*, *excludes* and *transfers*.

With these properties, roles can help a system effectively protect outside malicious messages from intruding the system.

In the current architecture of distributed systems, a computer affects other computers by issuing messages through the network interface. Other computers are affected through their network interfaces. The protection should be activated when a message is received. That is why a fire-wall is used. If the messages pass the checking of the fire-wall, the malicious codes in the messages still affect the target machines.

Under a RBAC system permissions are associated with roles [3]. Bertino et al. proposes an ID system to determine role intruders to database systems, that is, individuals that while holding a specific role, have a behavior different from the normal behavior of the role. They emphasize that RBAC mechanisms can be used to protect against insider threats.

In autonomic computing, a distributed system should be self-managing. A manager component should be able to detect exceptions at its network interface by checking the incoming messages including the packaged data and the message type.

From the message definition of E-CARGO, hacker messages basically simulate the regular or legal message pattern ℓ and hide malicious code in the parameter object p . To check whether a message has such malicious code is a typical task for protection mechanisms to accomplish. Traditional methods mainly involve using firewalls and anti-virus software. Firewalls are set up on an operating system and check the sending addresses of the incoming packages and block those packages from those addresses not allowed. Anti-virus software searches the files on the disks and checks the feature codes of viruses in files to state if a file has been infected by viruses.

Roles are good mechanisms for protecting the agent and the system from intrusion. With role-based approaches, an agent sends a message through its current role and then the message is dispatched by the role engine. In this situation, a malicious message can only be sent out by an agent with malicious aims.

Let us analyze the possible way to send out a malicious message to role-based systems. A hacker must pretend to issue a regular message m directing to a role r and put the malicious code into the parameter object p . Suppose that a role engine receives a message $m(r, p)$ from the hacker machine. It is the role r that decides the final destination of the message. The hacker will not be able to

send its malicious code to a specific destination. As stated before, role r is not easily infected because it has no code in it. Therefore, a hacker faces greater difficulty in attacking a role-based system versus traditional systems.

Furthermore, the role engine at the receiver side can check the message. This protection occurs when a message is received. A fire-wall like mechanism ($\mathcal{I}.f_m$) can be set in the role engine to check the source address of the message to protect messages from those malicious or unintended message senders.

$$\forall m \in \mathcal{M} (address(m) \in \mathcal{I}.f_m) \rightarrow dispatch(r, a, m, t_d) \quad \text{--- (9)}$$

If the message passes the first protect line- rule (9). It is still controlled and scrutinized. At the second protection line, it is the role that receives the message and dispatches its message to an agent. The activities of the agent are restricted by the role. The accessible objects are regulated by the role and its environment.

With role-based approaches, agents are set with a credit in a role engine to keep track of the agent's trust. If its trust cannot meet the requirement, no message will be dispatched to it and its role will be disapproved. If the malicious code wants to activate when the agent is scheduled to run, the roles could find the irregular behaviors of the agent and quickly tag it as an infected agent through rules (1-5).

7. SELF OPTIMIZATION

Optimization is often associated with space and time efficiencies. Optimizing a system infrastructure according to such specific objectives is complicated, because it is unclear at the beginning how to set relevant parameters which affect business objectives [1]. Moreover, optimizing the infrastructure is not a one time effort, as there may be changes which occur in the environment of the infrastructure, i.e., optimization must be dynamic and instant.

In a role-based system, different agents play different roles. The agents possess attributes related to space and time. Optimization jobs are activated when some qualified agents for a role are not best suited to play the current role. These agents may require additional space and time to complete tasks or services required by the role. The role engine could use idle times to instantly evaluate the performance of agents and transfer agents' roles to optimize the whole system. Assignment of potential agents to roles is a fundamental for dynamic optimization in role-based systems. The following rules (10-12) are used to help a role-based system do self-optimization.

For computation extensive systems,

$$\forall a \in \mathcal{A}, r \in \mathcal{R} (a.r_c \in r.\mathcal{A}_c) \rightarrow \neg (\exists a' \in r.\mathcal{A}_p \rightarrow a'.\alpha_a < a.\alpha_a) \quad \text{--- (10)}$$

For memory extensive systems,

$$\forall a \in \mathcal{A}, r \in \mathcal{R} (a.r_c \in r.\mathcal{A}_c) \rightarrow \neg (\exists a' \in r.\mathcal{A}_p \rightarrow a'.\beta_a < a.\beta_a) \quad \text{--- (11)}$$

For systems with balanced memory or computation requirement,

$$\forall a \in \mathcal{A}, r \in \mathcal{R} (a.r_c \in r.\mathcal{A}_c) \rightarrow \neg (\exists a' \in r.\mathcal{A}_p \rightarrow (a'.\alpha_a + a'.\beta_a) < (a.\alpha_a + a.\beta_a)) \quad \text{--- (12)}$$

8. RELATED WORK

Applying roles to offer autonomic properties is an innovative idea. However, very few writings mention such a topic. Some related articles discussed the adaptability of a system with the support of roles.

Colman and Han [5] discuss some ideas on applying roles into adaptive software development by elaborating how contracts can be used to monitor, regulate and configure the interactions between clusters of software entities called roles. By their method, abstract management contracts regulate the flow of control through the roles and provide monitoring interception points. Concrete contracts are domain specific and allow the specification of performance conditions. These contracts bind clusters of roles into self-managed composites—each composite with its own organizer role. The organizer roles can control, create, abrogate and reassign contracts. Adaptive systems are built from a recursive structure of such self-managed composites.

Tamai et al. [20] propose to realize object adaptation to environments with the support of a role-based model Epsilon and a language EpsilonJ. In Epsilon, an environment is defined as a field of collaboration between roles and an object adapts to the environment assuming one of the roles. Objects can freely enter or leave environments and belong to multiple environments at a time so that dynamic adaptation or evolution of objects is realized. Environments and roles are the first class constructs at runtime as well as at model description time so that separation of concerns is not only materialized as a static structure but also observed as behaviors. Environments encapsulating collaboration are independent reuse components to be deployed separately from objects.

Ren et al. [16] 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 in Fig. 1. In their model, behaviors of actors, roles and coordinators are formally defined and applied into supporting the re-configurability and fault localization for open distributed embedded software systems. The specified behaviors regulate some policies and regulations of agents' behaviors in a system, such as, memberships and configurations.

9. CONCLUSIONS

Self-* properties are pursued by the researchers of autonomic computing. This paper argues that roles and role-based approaches are good candidate solutions to support the self-* properties. Self-awareness, self-healing self-protection, and self-optimization are mainly discussed in this paper. In fact, roles can be taken as a good mechanism to facilitate all other self-* properties.

Newly introduced role-based mechanisms come at the cost of increased overhead in a system's performance. Accurate performance analyses are required to determine whether the benefits obtained by role-based approaches are worth the overhead introduced by role-based mechanisms.

ACKNOWLEDGMENTS

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

REFERENCES

- [1] Aiber, S., Gilat, D., Landau, A., Razinkov, N., Sela, A. and Wasserkrug, S., "Autonomic self-optimization according to business objectives", *Proceedings. International Conference on Autonomic Computing*, New York, NY, USA, 17-18 May 2004, pp. 206- 213.
- [2] Bertino, E., Bonatti, P. A. and Ferrari, E., "TRBAC: A Temporal Role-Based Access Control Model", *ACM Transactions on Information and System Security*, vol. 4, no. 3, Aug. 2001, pp. 191-223.
- [3] Bertino, E., Terzi, E., Kamra, A. and Vakali, A., "Intrusion detection in RBAC-administered databases", *Proc. of the 21st Annual Computer Security Applications Conference*, Dec. 5-9, 2005, pp. 170-182.
- [4] Bulusu, N., Heidemann, J., Estrin, D., Tran, T., "Self-configuring localization systems: Design and Experimental Evaluation", *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 1, Feb. 2004, pp. 24-60.
- [5] Colman, A. and Han, J., "Using role-based coordination to achieve software adaptability", *Science of Computer Programming*, vol. 64, 2007, pp. 223-245.
- [6] Covington, M. J., Moyer, M. J., and Ahamad, M., "Generalized Role-Based Access Control for Securing Future Applications", *23rd National Information Systems Security Conference*, 2000, avail: <http://smartech.gatech.edu/dspace/bitstream/1853/6580/1/GIT-CC-00-02.pdf>.
- [7] Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R. and Chandramouli, R., "Proposed NIST Standard: Role-Based Access Control", *ACM Transactions on Information and System Security*, vol. 4, no. 2, Aug. 2001, pp. 224-274.
- [8] Horn, P., "Autonomic Computing: IBM's perspective on the State of Information Technology", IBM Corp., Oct 2001, avail: <http://www.research.ibm.com/autonomic/>.
- [9] Joshi, J.B.D., Bertino, E., Latif, U., Ghafoor, A., "Generalized Temporal Role Based Access Control Model", *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 1, Jan. 2005, pp. 4-23.
- [10] Kephart, J. O., "Research Challenges of Autonomic Computing", *Proc. of the International Conference on Software Engineering (ICSE'05)*, May 15-21, 2005, St. Louis, Missouri, USA, pp. 15-22.
- [11] Lapouchnian, A., Yu, Y., Liaskos, S., Mylopoulos, J., "Requirements-Driven Design of Autonomic Application Software", *Proceedings of the conference of the Center for Advanced Studies on Collaborative Research Toronto*, ON, Canada, October 16 - 19, 2006, Article No. 7.
- [12] Lightstone, S., "Foundations of Autonomic Computing Development", *Proc. of the 4th IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASE'07)*, Tucson, Arizona, USA, 26-29 March 2007, pp. 163-171.
- [13] Müller, H., O'Brien, L., Klein, M. and Wood, B., "Autonomic Computing", *Technical Note*, CMU/SEI-2006-TN-006, Carnegie Mellon University, 2006, avail: <http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tn006.pdf>.
- [14] Nyanchama, M. and Osborn, S., "The Role Graph Model and Conflict of Interest", *ACM Trans. on Information and System Security*, vol. 2, no. 1, 1999, pp. 3-33.
- [15] Parashar, M., and Hariri, S., "Autonomic Computing: An Overview", UPP 2004, Mont Saint-Michel, France, *Banâtre, J.P., et al. (eds.), LNCS*, Springer Verlag, vol. 3566, 2005, pp. 247-259.
- [16] Ren, S., Yu, Y., Chen, N., Tsai, J. J.-P., and Kwiat, K., "The role of roles in supporting re-configurability and fault localizations for open distributed and embedded systems", *ACM Trans. Autonomic & Adaptive Sys.*, vol. 2, no. 3, Sept. 2007, pp.10:1-27.
- [17] Sandhu, R., Bhamidipati, V., Munawer, O., "The ARBAC97 Model for Role-Based Administration of Roles", *ACM Transactions on Information and System Security*, vol. 2, no. 1, Feb. 1999, pp 105-135.

- [18] Sterritt, R. Hinchey, M. "Engineering ultimate self-protection in autonomic agents for space exploration missions", *Proc. 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '05)*, 2005, pp. 506-511.
- [19] Sterritt, R. Hinchey, M. "Apoptosis and Self-Destruct: A Contribution to Autonomic Agents?" *Proc. Proceedings of the 3rd NASA/IEEE Workshop on Formal Approaches to Agent-Based Systems (FAABS2004)*, LNAI 3228, 2005, pp. 262-270.
- [20] Tamai, T., Ubayashi, N., and Ichiyama, R., "An Adaptive Object Model with Dynamic Role Binding", *Proc. of the Int'l Conf. of Software Engineering (ICSE'05)*, May 15-21, 2005, St. Louis, Missouri, USA, pp. 166-175.
- [21] Wang, J.F., *Cell Biology*, Science Press, Beijing, China, 2003 (in Chinese).
- [22] Zhu, H., "Role as Dynamics of Agents in Multi-Agent Systems", *System and Informatics Science Notes*, vol. 1, no. 2, July 2007, pp. 165-171.
- [23] Zhu, H., Grenier, M., Alkins, R., Lacarte, J. and Hoskins, J., "A Visualized Tool for Role Transfer", submitted to *The 6th Int'l Conf. on Infor. Sys. for Crisis Response and Management*, Washington, D.C., USA, May 4-7, 2008.
- [24] Zhu, H., "Fundamental Issues in the Design of a Role Engine", submitted to *the 2008 Workshop on Role-Based Collaboration, the 5th Int'l Symp. on Collaborative. Systems and Technologies*, San Diego, CA, USA, May 10-15, 2008.
- [25] Zhu, H. and Zhou, M.C., "Role Mechanisms in Information Systems - A Survey", *IEEE Trans. on Systems, Man and Cybernetics, Part C*, vol. 38, no. 6, June 2008 (in press), 20 pages.
- [26] Zhu, H. and Zhou, M.C., "Role-Based Collaboration and its Kernel Mechanisms", *IEEE Trans. on Systems, Man and Cybernetics, Part C*, vol. 36, no. 4, July 2006, pp. 578-589.
- [27] Zhu, H. and Zhou, M.C., "The Role Transferability in Emergency Management Systems", *Proc. of the 3rd Int'l Conf. on Info. Sys. for Crisis Response and Manag.(ISCRAM'06)*, Newark, USA, 2006, pp. 487-496.