Ontology-based Dynamic Role Interaction Control
in Multi-Agent Systems

Youna Jung
Advanced Computing and Information Systems (ACIS) Laboratory
P.O. Box 116200, 339E Larsen Hall, University of Florida, Gainesville, FL 32611-6200, USA
Tel: 352-392-1525, Fax: 352-392-5040
Email: younajung@ufl.edu

Minsoo Kim
LERSAIS, 410 IS Building, 135 N. Bellefield Avenue
Department of Information Science, University of Pittsburgh, PA 15260, USA
Email: danceletter@gmail.com

For more practical uses of multi-agent systems, the security on agents’ interactions is essential to allow only authorized interactions in multi-agent systems. In this paper, we propose a fine-grained Dynamic Role Interaction Access Control (DRiBAC) model to support exclusive authorization for an actual partnering agent of an interaction and dynamic control on the assignment of roles and permissions and their activation. To do this, we extend an earlier proposed RiBAC model with new concepts of role interaction, interaction assignment, and context-aware constraints. To describe the proposed model and its context and support semantic reasoning, we propose DRiBAC ontology. To verify the usability and efficiency of the ontology based DRiBAC, we implement a prototype system based on an example of an online tutoring system.

Keywords: Role Interaction, Mutual Exclusive Authorization, Dynamic Access Control, Context-Aware Constraint, Ontology-based Context Reasoning, MAS

ACM Classifications: D.4.6 Security and Protection-Access controls, I.2.4 Knowledge Representation Formalisms and Methods, I.2.11 Distributed Artificial Intelligence – Multiagent systems

1. INTRODUCTION

Recently, multi-agent technology has been considered as one of the most promising research areas (Wooldridge, 2009; Panait and Luke, 2005). An agent’s autonomous and dynamic problem solving behaviour has been the key to providing intelligent services in a variety of application areas such as online business, e-government, e-healthcare, etc. Moreover, the rapid growth of technologies related to networking and mobile devices has enabled rich social interactions among agents. Many multi-agent systems (MAS) have taken advantage of an agent’s ability to interact with other agents to solve complex and large-scale problems (Panait and Luke, 2005).

However, unauthorized interactions in MASs may cause serious security threats. For example, an agent may ask other agents to perform unsecure tasks or try to access a partner’s private resources. In addition, it may attempt to execute critical tasks on its partner agent like changing the partner’s status without appropriate authorization. We, therefore, need a suitable access control model to ensure that agents can engage in only authorized interactions to enable integrated, sharing activities.
We had earlier proposed the Role-interaction Based Access Control (RiBAC) model (Jung, Masoumzadeh, Joshi, and Kim, 2009), in which agents’ resources, tasks, and the agents themselves are also an entity to be protected as well as objects. Unfortunately, RiBAC does not capture the dynamic aspects of interactions among agents, which is critical in many MAS applications. RiBAC assumes that agents do not need to adjust a partnering agent or partner’s permissions. However, such an assumption is not practical in dynamic MASs. In reality, the agent’s context is continuously changing. Hence, we need to verify an agent’s qualifications for a partnering role and its permissions according to its changing context to ensure that its earlier authorization is still valid. Indeed, such context-aware behaviour is an agent’s basic characteristic (Wooldridge, 2009). Thus, it is important to dynamically authorize agents for various critical interactions and accesses based on changing context and behavioural information. Furthermore, RiBAC allows an interacting agent to have permissions for not only its partner agents but also other agents that are assigned to a partnering role. This means that an agent must share its properties such as its resources, context, and tasks with all the agents assigned to its partnering role even if its actual partner is only one agent – which is undesirable from the perspective of the principle of least privilege. While there are several advanced and context-aware access control models, RiBAC provides the administrative advantages of Role Based Access Control (RBAC) such as flexible, scalable, and abstraction-level administration (Sandhu and Munawer, 1999) and also addresses the interaction issue which is the key to MAS environments.

To remedy the shortcomings of RiBAC, in this paper, we propose the fine-grained Dynamic RiBAC (DRiBAC) model that captures actual interactions among agents and supports exclusive accesses only from their actual interacting agents. In addition, it allows dynamic assignments and adjustments based on context by using four types of context-based constraints: 1) assignment constraint, 2) activation constraint, 3) cardinality constraint, and 4) Separation of Duty (SoD) constraint. We also present ontology, written in OWL1.0, to describe the proposed model, its policies and a variety of context information. It provides the means for well-formed and reusable knowledge sharing among agents. To verify the proposed work, we also present a DRiBAC prototype system implemented for an online tutoring system.

The rest of this paper is organized as follows. In Section 2, we give an overview of the RiBAC model. Then, we present the limitations of RiBAC and propose the DRiBAC model as an extension of RiBAC with Role Interaction (RI), Interaction Assignment (IA), and the context-aware constraints. Next, we propose ontology to represent the DRiBAC model and its context in Section 4. To validate the proposed model and ontology, in Section 5, we present the implementation result of a DRiBAC prototype and two illustrative examples. Finally, in Section 6, we show some related work and in Section 7, we present our conclusion.

2. ROLE INTERACTION BASED ACCESS CONTROL (RiBAC)
In this section, we introduce the RiBAC model (Jung et al, 2009). RiBAC aims to protect not only objects but also agents’ tasks and agents themselves. To do so, in RiBAC, an interaction between agents is regarded as an access to a partnering agent or a partner’s task. In Figure 1, an overview of RiBAC is shown.

RiBAC is based on autonomous and goal-oriented agents who are the subjects that require appropriate permissions. Agents (AGENTS) are assigned to roles (ROLES) and can exercise the permissions assigned to the roles by activating them in a session (SESSIONS). In RiBAC, each role has a set of permissions (PERMISSIONS), which include object-oriented permissions (OPRMS) and two types of interaction permissions: role-oriented permissions (RPRMS) and task-oriented
permission (TPRMS). OPRMS is the same as the permissions in RBAC, which is represented as a pair that includes an operation and an object.

To handle interaction among agents, RiBAC defines two types of role interactions: Role-Oriented (RO) interaction and Task-Oriented (TO) interaction. An ro interaction indicates that a subject role ($R_s$), starting an interaction, performs its task on a targeted object role ($R_o$), while a to interaction indicates that $R_s$ commands $R_o$ to perform a task of $R_o$. For example, let’s assume an emergency situation in an agent-based environment where everyone has a personal smart device in which an agent acting a person in a social network resides. A man collapses in the street and paramedics come to help him. A paramedic can transfer a patient to a hospital. Such interaction between a paramedic and a patient is a ro interaction that the paramedic role is $R_s$, and the patient role is a $R_o$. In addition, a paramedic can request the first-aid instructions for a patient from a doctor who is in charge of the patient. The interaction between the paramedic role and the doctor role is a to interaction. The paramedic role is $R_s$ and its $R_o$ is the doctor role, which should provide a first-aid instruction to a paramedic. For a successful role interaction, $R_s$ must have corresponding interaction permissions and its $R_o$ should have all the permissions necessary for accomplishing the requested tasks.

To get a permission, each agent should take a role that assigned the permission through the Permission Assignment (PA) relationship. A valid pairing of an object and an operation on that object forms an oprms. Roles are authorized for object-oriented permissions that are assigned to them through the Object-oriented Permission Assignment (OPA). A valid pair of $R_s$’s tasks and $R_o$ forms an rprms, and it is assigned through the Role-oriented Permission Assignment (RPA). A tprms consists of $R_s$ and $R_o$’s task, which can be invoked by other roles, and it is assigned to $R_s$ through the Task-oriented Permission Assignment (TPA).

3. DYNAMIC RiBAC (DRiBAC)

3.1 Motivation
A key limitation of RiBAC is that there is no way to distinguish agents assigned to the same interacting role. Thus, an agent, say $A$, has permissions for all the agents assigned to its partnering role, even though most of these agents may not be related to $A$. To avoid such oversharing and guarantee the principle of least privilege, we would need to create a new interacting role for the agent. However, this approach will increase the number of roles needed to capture all such needs. In the worst case, the number of roles exceeds the number of agents. It means that we cannot take advantage of RBAC.

Furthermore, RiBAC does not consider the dynamic nature of agents. It handles agent’s interactions in a static manner, regardless of the agent’s context-awareness and dynamic behaviour. In RiBAC, an administrator (or, administrative agent) manually assigns roles to agents and there is no way to adjust the assigned permissions according to the changes in context. However, it is risky
to allow an agent to have permissions when the agent’s context changes and it becomes ineligible for earlier interaction permissions.

3.2 DRiBAC Specification
To remove such limitations, we need an access control model that captures actual interaction relationships among agents and supports an agent’s dynamic behaviour. We therefore propose the fine-grained Dynamic RiBAC (DRiBAC) model. DRiBAC has all the benefits of RBAC and also provides dynamic and fine-grained control on interactions among agents using context at the same time. In the following section, we explain two key contributions of DRiBAC in detail.

3.2.1 Role Interaction and Interaction Assignment
DRiBAC allows interacting agents to give permissions only to a partnering agent among agents taking the same role by using new concepts of the role interaction (RI) and the Interaction Assignment (IA) relationships. The overview of DRiBAC is shown in Figure 2.

Similar to RiBAC, the proposed model is based on interacting agents, which have autonomous, goal-oriented, and context-aware behaviour. Each agent has its own properties including resources, context, and tasks. An agent’s context represents the status of both its resources and tasks. In DRiBAC, a relationship between two interacting roles is specified as a role interaction (RI). Here, we assume that all interactions are on a one-to-one basis. DRiBAC deals with one-to-many and many-to-many relations by splitting them into several one-to-one interactions. Two interacting roles in an ri are assigned to two actual agents by an interaction assignment (IA). Using RI and IA, DRiBAC specifies the interaction relations between two actual agents and provides exclusive authorization for a partnering agent only. To control accesses to an agent’s properties, DRiBAC defines three types of interaction permissions: resource-oriented permission (SPRMS), role-oriented permission (RPRMS), and task-oriented permission (TPRMS). Compared to RiBAC, DRiBAC additionally supports the access control on agents’ resources, while RiBAC does not consider them. Here, we note that DRiBAC policies related to an agent’s resources are enforced by the agent. To access an agent’s resource, others should ask the agent having the resource for the permission. On the other hand, a system’s resources, represented as objects in DRiBAC, belong to a system so they are controlled by a system administrator.

As two interacting roles of an ri are assigned to two agents (IA) by an administrator, each agent has corresponding interaction permissions for its partnering agent and an interaction between two
agents is actually started. Agents must notify an administrator of the end of their interaction as soon as they finish the interaction. Then the administrator re-assigns the assigned interaction (IA) from two agents. In DRiBAC, we assume that all agents in a MAS are not malicious, thus we do not consider any false notification from malicious agents. For precise understanding and use of DRiBAC, we present the formal specification in Table 1.

### 3.2.2 Context-Aware Dynamic Constraints

In DRiBAC, roles, permissions, and role interactions are dynamically assigned to agents based on context information, which includes not only general context, such as time and location, but also the agents’ context at runtime. Consequently, only qualified agents are assigned to a role and keep playing that role until the end of an interaction. To prevent agents from the misuse of their properties, an agent’s roles, permissions, and interactive relations can be deactivated during interactions, if they are disqualified by a change in context. To do this, DRiBAC uses four types of

---

Table 1: Formal specification of DRiBAC

<table>
<thead>
<tr>
<th>Description</th>
<th>Notation</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENTS(A): the set of all agents in a system, an agent a has its own resources, context, and tasks.</td>
<td>$AGENTS(A)$</td>
<td>$a = (RSC_a, CONT_a, TSK_a)$; $RSC_a \in RSC$ is a set of resources which an agent a has, $CONT_a \in CONT$ is a set of context which an agent a has, and $TSK_a \in TSK$ is a set of the agents’ tasks.</td>
</tr>
<tr>
<td>$RSC = { \forall a \in A \mid RSC_a }$, the set of resources which agents have.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$CONT = { \forall a \in A \mid CONT_a }$, the set of all possible context which represent agents.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$TSK$: the set of all tasks that agents can perform.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$OPS \in TSK$: the set of all operations of roles on OBJ.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$ROLES(R)$: the set of all roles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$ROLE INTERACTIONS(RI) = R \times R$: the set of two interacting roles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$PERMISSIONS(P) = OPRMS \cup iPMS$, a set of permissions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$OPRMS \subseteq OPS \times OBJ$ where OBJ is a set of objects which belong to a system, same as OBJ in RBAC’96, the set of all object-oriented permissions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$OBJ = OBJ \cup OBJ-RSC \cup OBJ-ROLE \cup OBJ-TSK$ where OBJ-RSC $\subseteq$ RSC, OBJ-ROLE $\subseteq$ R, and OBJ-TSK $\subseteq$ TSK, the set of all target objects of OPS and it can be a system’s object, an agent’s resource or task, or the role itself.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$INTERACTION PERMISSIONS (iPRMS) = SPRMS \cup RPRMS \cup TPRMS$, a set of interaction permissions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$SPRMS \subseteq OPS \times OBJ-RSC$, the set of resource-oriented permissions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$RPRMS \subseteq OPS \times OBJ-ROLE$, the set of role-oriented permissions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$TPRMS \subseteq OPS \times OBJ-TSK$, the set of task-oriented permissions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$AA \subseteq A \times R \times (C)$ where C is a context-aware constraint and an optional one, a many-to-many role to agent assignment relation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$INTERACTION ASSIGNMENT (IA) \subseteq AA \times AA \times C$, many-to-many interacting role to agent assignment relation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$PA \subseteq R \times P \times (C) = { OPA \cup iP } \times (C)$ where $iPA = { SPA \cup GPA \cup TPA }$, many-to-many roles to permissions assignment relation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$OPA \subseteq R \times OPRMS$, $SPA \subseteq R \times SPRMS$, $RPA \subseteq R \times RPRMS$, $TPA \subseteq R \times TPRMS$.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$SESSIONS(S)$: the set of all sessions created for agents.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Journal of Research and Practice in Information Technology, Vol. 43, No. 3, August 2011

213
context-aware constraints imposed on the assignment and the activation of roles, permissions, and interactions: Assignment Constraint, Activation Constraint, Cardinality Constraint, and SoD Constraint. We present the expression of constraints in Table 2 and describe each of them in detail.

- **Assignment Constraints** – represent the context-based conditions for assigning roles, permissions, and interactions to agents. Here, the conditions in assignment constraints mean the minimum qualifications of agents to be assigned. If an assignment condition is related to the agents’ context, a system can recommend the candidates for the assignments automatically. If there are more than

<table>
<thead>
<tr>
<th>Categories</th>
<th>Constraints</th>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment Constraint</td>
<td>AA</td>
<td>assignaa(a, r, c)</td>
<td>If context-aware constraint c is satisfied, an agent a is assigned to a role r.</td>
</tr>
<tr>
<td></td>
<td>PA</td>
<td>assignpa(p, r, c)</td>
<td>If c is satisfied, a permission p is assigned to r.</td>
</tr>
<tr>
<td></td>
<td>IA</td>
<td>assignia(ai, ri, rj, cj) &amp; assignia(aj, ri, rj, cj)</td>
<td>Where ri and rj are associated with a role interaction ri, if cj is satisfied, ai is assigned to ri and if cj is satisfied, aj is assigned to rj.</td>
</tr>
<tr>
<td>Activation Constraint</td>
<td>AA</td>
<td>activate,assignaa, c</td>
<td>If c is satisfied, a can activate r.</td>
</tr>
<tr>
<td></td>
<td>PA</td>
<td>activate,assignpa, c</td>
<td>If c is satisfied, r can acquire p.</td>
</tr>
<tr>
<td></td>
<td>IA</td>
<td>activate,assignia, ci &amp; activate(assignia, cj)</td>
<td>If ci and cj are satisfied, ai and aj can interact with each other.</td>
</tr>
<tr>
<td>Static Cardinality Constraint</td>
<td>Total No. of IA</td>
<td>(|IA</td>
<td>, Nassign, c)</td>
</tr>
<tr>
<td></td>
<td>Per agent</td>
<td>(|RI.IA</td>
<td>, Nassign, c)</td>
</tr>
<tr>
<td></td>
<td>Per RI-agent</td>
<td>(|RI.IAa</td>
<td>, Nassign, c)</td>
</tr>
<tr>
<td>Dynamic Cardinality Constraint</td>
<td>Total No. of active IA</td>
<td>(|IAactive</td>
<td>, Nactive, c)</td>
</tr>
<tr>
<td></td>
<td>Per agent</td>
<td>(|RI.IAactive</td>
<td>, Nactive, c)</td>
</tr>
<tr>
<td></td>
<td>Per RI-agent</td>
<td>(|RI.IAactivea</td>
<td>, Nactive, c)</td>
</tr>
</tbody>
</table>

| SoD | SSoD | Exclusive RIs | For ri \leftrightarrow \rightarrow rj, a \rightarrow \rightarrow rj, IAa | a cannot be assigned to exclusive RIs |
| | Excl. Roles | For rj, assign(a, rj) \leftrightarrow assign(a, rj) | a cannot be assigned to exclusive roles |
| | DSoD | Exclusive RIs | For ri \leftrightarrow \rightarrow rj, rj, IAa \rightarrow activate \leftrightarrow rj, IAa | a cannot activate exclusive RIs at the same time |
| | Excl. Roles | For rj, active(a, rj) \leftrightarrow active(a, rj) | a cannot activate exclusive roles at the same time |

Table 2: Context-Aware Constraints for DRIBAC
two agents who satisfy the corresponding condition, an administrator should select the most suitable one. For an IA, we need to perform agent assignments (AA) twice for an RI’s two interacting roles, ri.rj and ri.rj. If the agent is disqualified due to the change of its context later, the assigned role, permission, and interaction are automatically de-assigned for security.

- **Activation Constraints** – To activate the assignment relations, the corresponding context-aware constraints should be checked. If the conditions are not satisfied, the assignment relations cannot be activated and the assignments activated once are automatically deactivated. During interaction, an interacting agent assigned to an interacting role of an RI is deactivated; the corresponding interaction is temporarily deactivated until all of the interacting agents are to be activated. However, some of assignment relationships may not require the activation constraints. In such a case, the assignment is immediately activated after assigning.

- **Cardinality Constraints** – In DRiBAC, we specify the maximum cardinality constraints by limiting the number of interaction assignments (IA). In detail, we can limit the total number of IAs in a system, an RI, an agent, or an agent of an ri. The cardinality constraints are divided into two types: *Static Cardinality Constraint* and *Dynamic Cardinality Constraint*. The static one is applied on the assignment of interactions among agents, while a dynamic one is used on the activation of interactions.

- **SoD Constraints** – DRiBAC has SoD Constraints to handle the problems of mutually exclusive roles and exclusive role interactions. For exclusive role interactions, expressed ri↔ri, DRiBAC supports static SoD (SSoD) and dynamic SoD (DSoD) constraints. The SSoD constraints for exclusive RIs prevent agents from assigning to interactions of exclusive RIs at the same time (ri.IAa ↔ ri.IAa), while the DSoD constraints prevent simultaneous activations of those interaction relations (ri.IAaactive ↔ ri.IAaactive). For exclusive roles, expressed ri↔ri, DRiBAC supports SSoD and DSoD constraints to avoid assigning/activating mutually exclusive roles at the same time.

### 3.3 Example: Online Tutoring System with DRiBAC

In this section, we present how DRiBAC works for secure interactions among agents and also overcomes the drawbacks of RiBAC. To do this, we apply the model to an online tutoring system shown in Figure 3. This system connects students to tutors based on students’ context, such as evaluation results or preferences, and provides several services to support online tutoring, such as administrative services on resources and context in the system like tutors’ teaching materials and students’ evaluation results; a matching service between tutors and students, and tutoring services like online video tutoring using web cameras. We assume that there are two roles, *Tutor* and *Student*, in the system and many people are assigned to the two roles.

#### 3.3.1 Mutual Exclusive Interaction and Assignment Constraint

Assume that Julie, a student, wants to give permissions for her properties, resources, context, and tasks only to Anna, her tutor. To do this, in RiBAC, we should create new roles for only two agents, Julie and Anna. However, in DRiBAC, we can administer the required interaction without further modification of the earlier defined roles by using the corresponding RI and IA. In addition, if Julie has a specific requirement that her tutor should satisfy, for example: "My tutor must be female and the number of her students must be no more than three". To fulfill her request, we can specify a context-based constraint on the interaction assignment for Julie and her tutor in DRiBAC shown as follows:

\[
C_{Julie's tutor} = T.student\_Number \leq 3 \& T.sex=Female
\]
Based on the suggested constraint, a system can recommend the candidates for Julie’s tutor and assign Anna automatically. If there are more than two candidates satisfying a constraint, an administrator can manually assign roles, permissions, and interactions based on the system’s recommendation. If Anna is disqualified later by the change in her context, her role as Julie’s tutor is automatically deactivated and the interaction between them is deactivated accordingly.

$$RI_{1} = (T, S) & IA = \{(T:Anna, T.student\_Number \leq 3 & T.sex=Female), (S:Julie)\}$$

### 3.3.2 Activation Constraints

Let’s assume that Anna has three weeks of maternity leave during the tutoring period. In this situation, it is safe to restrict her permissions for Julie’s properties. In addition, it is better to know the availability of the interaction permissions that an agent has. For example, Anna can avoid assigning homework that is beyond Julie’s capability if she knows Julie’s progress in class work. In addition, Julie won’t request an online video tutoring session for Anna if she knows that Anna is on maternity leave.

Unlike in RiBAC, in DRiBAC, the interaction permissions that interacting agents have are continuously adjusted according to the changes in context. Using the context-based activation constraints of DRiBAC, we can guarantee the principle of least privilege more strictly than RiBAC. For example, the rprms to evaluate Julie’s classwork is assigned to Anna but it won’t be activated unless Anna continues her tutor work for over one month. Such an activation constraint can be expressed as follows:

$$rprms_{1} = \{\text{Evaluating\_Student, S}\} & \text{assign}_{ia1}(rprms_{1}, T) & \text{activate}(\text{assign}_{1}, \text{Training\_Period} \geq 1\text{month})$$

If Julie’s tprms is deactivated based on Anna’s context like ‘on maternity leave’, Julie would not waste her time and resources and would seek out another available tutor. For such dynamic adjustment of IAs, we can specify corresponding IA as follows:

$$tprms_{1} = \{\text{request, T.Live\_Tutoring}\} & \text{assign}_{ia2}(tprms_{1}, S) & \text{activate}_{ia}(\text{assign}_{ia2}, (T.Online\_Status=Online \& T.Live\_Tutoring\_Status=Available))$$
3.3.3 Cardinality Constraints
In this tutoring system, there are several membership plans for students. According to the membership, the number of available tutors differs. For example, if a student has the gold membership, he/she can have five tutors. Let’s assume that every gold member can activate four tutoring interactions at most and Julie, a gold member, is fully activating them. In this situation, Julie cannot have a chance to get a new tutor. However, if the interaction with Anna is deactivated, she can interact with a new tutor. In DRiBAC, we perform such complex administrative work easily using a dynamic cardinality constraint shown as follows:

\((|RI_1,IA_a|, 5, a.membership=Gold) \& (|RI_1,IA_a^{active}|, 4, a.membership=Gold)\)

If the system must restrict the number of interactions between tutor and student at daytime due to the limitation of bandwidth, we should restrict the number of all active interactions of RI with the change of time as follows:

\((|RI_1,IA^{active}|, 12 millions, time=Daytime) \& (|RI_1,IA^{active}|, 15 millions, time=Nighttime)\)

3.3.4 SoD Constraints
For a tutoring interaction, RI, we should avoid assigning an agent to two exclusive roles of RI, T and S. Such a constraint can be expressed as follows: \(assign_{ia}(a, T) \leftrightarrow assign_{ia}(a, S)\). In addition, assume that a tutor is assigned to two types of interactions, a tutoring interaction (RI) and an exam writing interaction (RI2). For the security of an exam, it is prohibited that an exam writer (E) has contact with his/her students until the exam is finished. In such a case, the tutoring interaction and the exam writing interaction are mutually exclusive. In DRiBAC, we specify such a constraint as a DSoD constraint as follows: \(RI_2=(E, S) \& RI_1,IA_{a}^{active} \leftrightarrow RI_2,IA_{a}^{active}\)

4. ONTOLOGY FOR DRiBAC
Using ontology, a model can be semantically specified, since ontology is an explicit and well-formed specification of shared knowledge. In this paper, we thus define DRiBAC ontology, shown in Figure 4, addressing following issues:
 Ontology-based Dynamic Role Interaction Control in Multi-Agent Systems

• Explicit Description of DRiBAC Model – Using ontology, we can specify DRiBAC explicitly. All the concepts in DRiBAC such as agents’ interactions and context-aware constraints can be formalized as concepts and relationships in DRiBAC ontology.

• Formal Language for Policy Description – Besides the entities related to the model such as role, permission, and interaction, the policies of DRiBAC can be described. It means that DRiBAC ontology can be used as a formal policy language for DRiBAC as well as a method to describe the model itself.

• Dynamic Access Control for Interaction among Agents – Using concepts and relations in the context ontology, we can describe rich context information including the agents’ context and dynamic constraints based on context in a well-formed way. Context information in DRiBAC is dynamically updated by interaction with context ontology. To evaluate the incoming access requests, DRiBAC ontology should access the corresponding context ontology.

4.1 RBAC Entities
To build the foundation of DRiBAC ontology, we define several classes related to RBAC entities: Role, User, Permission, and Session. According to RBAC, we define each of them with their own properties as shown in Table 3.

<table>
<thead>
<tr>
<th>Class</th>
<th>Definition</th>
</tr>
</thead>
</table>
| dribac:Role | rdfs:subClassOf owl:Thing  
  dribac:assignedUser a owl:ObjectProperty  
  rdfs:domain dribac:Role; rdfs:range : dribac:Agent.  
  dribac:hasPermission a owl:ObjectProperty  
  rdfs:domain dribac:Role; rdfs:range dribac:Permission |
| dribac:User | rdfs:subClassof owl:Thing  
  owl:equivalentClass dribac:Agent  
  dribac:assignedRole a owl:ObjectProperty  
  rdfs:domain dribac:User; rdfs:range : dribac:Role |
| dribac:Permission | rdfs:subClassOf owl:Thing  
  dribac:assignedRole a owl:ObjectProperty  
  rdfs:domain dribac:Permission; rdfs:range dribac:Role |
| dribac:Session | rdfs:subClassof owl:Thing  
  dribac:hasSessionRole a owl:ObjectProperty  
  rdfs:domain dribac:Session; rdfs:range dribac:Role  
  dribac:hasSessionUser a owl:ObjectProperty  
  rdfs:domain dribac:Session; rdfs:range dribac:User |

Table 3: RBAC Entities of DRiBAC Ontology

4.2 DRiBAC Entities
To represent DRiBAC completely, the proposed ontology needs to describe new concepts such as the agent’s properties, role interactions, and agent interactions. First of all, the class Agent has four properties, agentID, hasResource, hasContext, and hasTask. This class is defined as an equivalent Class to the Class User of RBAC, because agents are the subjects that have all permissions, the same as users in RBAC.

To describe interaction permissions, DRiBAC ontology has the InteractionPermission class and three sub classes: ObjectOrientedPermission, RoleOrientedPermission, and TaskOrientedPermission. Each of them is defined as a subclass of not only Permission class but also InteractionPermission class to maintain consistency between RBAC and DRiBAC. The classes have two special properties, hasObjectRole and hasSubjectRole.

For specifying DRiBAC constraints, four classes are defined: AssignmentConstraint, ActivationConstraint, CardinalityConstraint, and SoDConstraint. These classes include context-based condition that is represented by using classes and properties within context ontology.
4.3 Context Ontology
It is not easy to define context ontology, since there are incredibly diverse types of context and the required context information differs depending on the applications’ domains. A few general context ontologies have been proposed in the literature, such as SOUPA (Chen, Finin and Joshi, 2005), GAS (Christopoulou and Kameas, 2005), and Context (Context, 2009). They are upper ontologies, thus users should modify them for use in a specific system. In this paper, we define the context ontology by employing the classes of SOUPA and Context and extending them for the online tutoring system example. Figure 5 illustrates a portion of the context ontology developed.

5. DRiBAC ENFORCEMENT
In this section, we describe how to dynamically enforce DRiBAC policies using the DRiBAC ontology and the context ontology. To build those ontologies, we use protégé, an ontology editor and engineering tool developed by Stanford University Medical Center. In addition, we use SPARQL to query the ontologies.

5.1 Predicates and Axioms for Policy Enforcement
Prior to presenting constraints rules and axioms, we introduce several system functions used for defining rules. The assigns() function performs AA, PA, and IA, while the activates() function activates the assignment relations. For an IA, the assigns(a, ri.r) function is performed twice for ri.r and ri.rj ((ri,rj)∈ri). The assigned(), can_be_assigned(), and activated() functions are a Boolean function to examine assignment/activation relationships. The isSatisfied(condition) function evaluates context condition by reasoning on context ontology and numberOf(class) calculates the number of individuals in a particular class.

Table 4 shows predicates and axioms for assignment constraints. If isSatisfied (assignment_condition) returns true, an administrator can perform corresponding assignment. For
Ontology-based Dynamic Role Interaction Control in Multi-Agent Systems

Table 4: Assignment Constraints

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.assignedRole(r) ∧ r.assignedAgent(a) → assigned(a, r)</td>
<td>a has assigned role r and r has assigned agent a.</td>
</tr>
<tr>
<td>p.assignedRole(r) ∧ r.assignedPermission(p) → assigned(p, r)</td>
<td>p has assigned role r and r has assigned permission p.</td>
</tr>
<tr>
<td>¬assigned(a, r) ∧ isSatisfied(c) → assigns(a, r)</td>
<td>If assignment condition c is satisfied, agent a is assigned to role r.</td>
</tr>
<tr>
<td>¬assigned(p, r) ∧ isSatisfied(c) → assigns(p, r)</td>
<td>If assignment condition c is satisfied, permission p is assigned to role r.</td>
</tr>
<tr>
<td>assigned(a, r) ∧ isSatisfied(c') → deassigns(a, r)</td>
<td>If assignment condition c' is satisfied, agent a is deassigned from role r.</td>
</tr>
<tr>
<td>assigned(p, r) ∧ isSatisfied(c') → deassigns(p, r)</td>
<td>If assignment condition c' is satisfied, permission p is deassigned from role r.</td>
</tr>
<tr>
<td>assigned(a1, r1, r1i) ∧ assigned(a2, r2, r1i) ∧ r1.involved(r1i) and r2.involved(r1i) → can_be_assigned(a1, r1, r1i) ∧ can_be_assigned(a2, r2, r1i)</td>
<td>If assignment conditions are satisfied for both agents a1 and a2 to role r1 with interaction r1i, then they can be assigned.</td>
</tr>
<tr>
<td>can_be_assigned(a1, r1, r1i) ∧ can_be_assigned(a2, r2, r1i) ∧ isSatisfied(c1) ∧ isSatisfied(c2) → assigns(a1, r1, r1i) ∧ assigns(a2, r2, r1i)</td>
<td>If assignment conditions c1 and c2 are satisfied for agents a1 and a2 to role r1 with interaction r1i, they are assigned.</td>
</tr>
</tbody>
</table>

Table 5: Activation Constraints

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assigned(a, r) ∧ isSatisfied(c) → activate(a, r)</td>
<td>If assignment condition c is satisfied, agent a is activated for role r.</td>
</tr>
<tr>
<td>assigned(p, r) ∧ isSatisfied(c) → activate(p, r)</td>
<td>If assignment condition c is satisfied, permission p is activated for role r.</td>
</tr>
<tr>
<td>activated(a, r) ∧ isSatisfied(c') → deactivate(a, r)</td>
<td>If activation condition c' is satisfied, agent a is deactivated from role r.</td>
</tr>
<tr>
<td>activated(p, r) ∧ isSatisfied(c') → deactivate(p, r)</td>
<td>If activation condition c' is satisfied, permission p is deactivated from role r.</td>
</tr>
<tr>
<td>assigns(a1, r1, r1i) ∧ assigns(a2, r2, r1i) ∧ isSatisfied(c) → activate(a1, r1, r1i) ∧ activate(a2, r2, r1i)</td>
<td>If assignment condition c is satisfied, agents a1 and a2 are activated for role r1 with interaction r1i.</td>
</tr>
<tr>
<td>deactivate(a1, r1, r1i) ∧ deactivate(a2, r2, r1i) → deactivate(a1, r1, r1i) ∧ deactivate(a2, r2, r1i)</td>
<td>If agents a1 and a2 are deactivated from role r1 with interaction r1i, then they are deactivated.</td>
</tr>
<tr>
<td>(activate(a1, r1, r1i) ∧ isSatisfied(c1')) ∨ (activate(a2, r2, r1i) ∧ isSatisfied(c2')) → deactivate(a1, r1, r1i) ∧ deactivate(a2, r1, r1i)</td>
<td>If either activation condition c1' or c2' is satisfied for agents a1 or a2 to role r1 with interaction r1i, then they are deactivated.</td>
</tr>
</tbody>
</table>

IAs, two agents can be assigned to an RI where they are already assigned to each role of that RI. When two agents can be assigned and isSatisfied (assignment_condition) returns true, then the system assigns the agents to the interaction.

Similar to the assignment constraints, activation rules are defined as shown in Table 5. Note that when one agent participating in an interaction is deactivated, then the interaction is deactivated.

The static cardinality constraints are checked by counting individuals of Interaction classes. Since an individual is created when an IA is enforced and removed when the interaction is ended, total number of IAs is the same as the number of individuals. The static cardinalities per RI and per agent are restricted by using the properties of Interaction Class. The per RI-Agent cardinality is restricted by using owl:sameAs keyword between per RI and per Agent individuals. It means that we figure out particular IAs in which agent a participates among all IAs of an RI.

The dynamic cardinality constraints are examined in similar way. The total number of active IAs can be obtained by counting Individuals of Interaction Class whose ‘activated’ property is set as true. This property is automatically set when the rule described in Table 6 returns true. The other dynamic cardinality constraints are restricted by using owl:sameAs between the assigned interactions and the activated interactions.

To represent SoD rules, the ‘exclusiveRI’ property of RoleInteraction class and ‘exclusiveRole’ property of Role class are used as shown in Table 7. Those properties are defined as owl:SymmetricProperty.
5.2 System Architecture
The dynamic policy enforcement in a DRiBAC system is presented in Figure 6. We assume that all agents and the DRiBAC engine share the ontology repository including context ontology and DRiBAC ontology and the SPARQL query execution engine. When the Policy Enforcement Engine (PEE) receives a request for access permissions from an agent or from the DRiBAC engine (1), it delivers the request to the Policy Decision Engine (PDE). PDE then generates a SPARQL query by interpreting the request using corresponding policies stored in the policy repository (2). The created query is sent to the SPARQL execution engine (3), and then the engine executes the query on the

![Figure 6: Enforcement Flow on DRiBAC System](image)

<table>
<thead>
<tr>
<th><strong>Static Cardinality Constraints</strong></th>
<th><strong>Dynamic Cardinality Constraints</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total No. numberOf(Interaction) ≤ Nassigned</td>
<td>numberOf(Interaction.activated) ≤ Nactivated</td>
</tr>
<tr>
<td>per RI numberOf(Interaction.associated-RoleInteraction(ri)) ≤ Nriassigned</td>
<td>numberOf(Interaction.activated owl:sameAs Interaction.associatedRoleInteraction(ri)) ≤ Nactiv</td>
</tr>
<tr>
<td>per Agent numberOf(Interaction.activated owl:sameAs Interaction.participants(a)) = Nn</td>
<td>numberOf(Interaction.activated owl:sameAs Interaction.participants(a)) = Nnactiv</td>
</tr>
<tr>
<td>per RI-Agent numberOf(Interaction.associated-RoleInteraction(ri) owl:sameAs Interaction.participants(a)) ≤ Nri.a</td>
<td>numberOf(Interaction.activated owl:sameAs Interaction.participants(a)) = Nri.aactiv</td>
</tr>
</tbody>
</table>

Table 6: Cardinality Constraints

<table>
<thead>
<tr>
<th><strong>SSoD</strong></th>
<th><strong>DSoD</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclusive RIs</td>
<td>r_i,exclusiveRI(ri) ∧ ia,associatedRI(ri) → ¬assigns(a,ri)</td>
</tr>
<tr>
<td></td>
<td>i,associatedRI(ri) ∧ activated(a,ri) → ¬activates(a,ri)</td>
</tr>
<tr>
<td>Exclusive Roles</td>
<td>r_i,exclusiveRole(rj) ∧ assigned(a,ri) → ¬assigns(a,ri)</td>
</tr>
<tr>
<td></td>
<td>r_i,exclusiveRole(rj) ∧ activated(a,ri) ¬activates(a,ri)</td>
</tr>
</tbody>
</table>

Table 7: SoD Constraints
ontology reasoner that accesses the ontology repository (4). After the ontology reasoner infers from existing context and DRiBAC policies, the reasoning result responds to the PDE (5). If it is required to check SoD or Cardinality constraints, PDE generates another query for it and repeats steps (3)-(5). After PDE generates a decision of authorization, PEE updates the ontology repository to reflect the changed information (6). Finally, the decision is enforced by PEE (7).

5.3 Running Example
In this section, we describe a running example based on the example scenario shown in Section 3.3.

1. **Agent Assignment:** In DRiBAC, a role is dynamically assigned to an agent, whose context condition satisfies the assignment condition of the role. For example, to take the Tutor role, an agent should possess particular resources, such as *Teaching Evaluation* and *Teaching Materials*, and context data such as *online teaching availability* and the *number of students*. In addition, it should be capable of doing some tasks like *onlineTeaching* and *EvaluatingStudent*. All context-aware constraints, including assignment constraints, are expressed as a SPARQL query as illustrated in Figure 7(a). The SPARQL queries contain names of classes and properties in the ontology repository.

2. **Interaction Assignment:** To avoid unsecure interactions with unqualified agents, context-aware constraints can be specified for IAs. For example, Julie’s requirement for her tutor depicted in Section 3.3 can be specified as a SPARQL query which is shown in Figure 7(b). As you see, there are three tutors who can do online tutoring (Figure 7(a)), but Anna is finally assigned to Julie’s tutor (Figure 7(b)).

3. **Dynamic permission adjustment:** During interaction, interacting agents should control the accesses to their own properties depending on the change in context. Let’s remember the constraint for assigning homework to students described in Section 3.3. The condition can be expressed as ‘If the student’s classwork progress is over 80% and the attendance rate is over 75% then the tutor can assign homework’. The SPARQL query incorporating the condition and its result is shown in Figure 7(c).

![Figure 7: Enforcement on DRiBAC System](image-url)

---

**Figure 7:** Enforcement on DRiBAC System
6. RELATED WORK

Several models that exploit security relevant information of users, roles, and environments have been proposed in the literature. Al-Kahtani and Sandhu (2002) proposed an attribute-based access control model (ABAC) to enforce User-Role assignment (UA) dynamically. The model defines rules for UA by using name-value pairs of attributes. It performs UA dynamically only when the rules are satisfied at runtime. Rule based RBAC (RB-RBAC) has been introduced to improve the expressiveness of ABAC (Kern and Walhron, 2005). It supports automatic UA using a finite set of authorization rules. Several other context/situation-based RBAC models have been proposed that incorporate context constraints to support dynamic access control (Hulsebosch, Salden, Bargh, Ebben and Reitsma, 2005; Peleg, Beimel, Dori and Denekamp, 2008; Kim, Joshi and Kim, 2009).

Recently, there have been some efforts to use OWL as a representation language for RBAC policies. Finin et al (2008) have introduced ROWLBAC, a representation of RBAC in OWL. Knetchel et al (2008) proposed an approach that uses OWL for reasoning about RBAC authorizations. The model supports both roles and class hierarchies. However, it does not take into consideration the SoD constraints. Di et al (2005) proposed an approach that uses OWL to specify the RBAC constraints. However, to specify the SoD and other constraints, additional rules must be required. Kagal et al (2006) proposed a general framework using semantic web technologies to support general purpose policy systems and resolve mismatches among different policy languages. Cirio et al (2007) proposed an access control system using OWL and DL for context-awareness. They developed simple context ontology to capture the features of a sample scenario.

Since some existing works focused on RBAC-based ontologies, they cannot deal with the role-interaction and interaction assignment that are the core of the DRiBAC model. Furthermore, they do not consider incorporation between access control ontology and context ontology even though context information is essential to provide dynamic access control.

In this paper, we define ontology for not only representing DRiBAC model in a well-formed way but also for expressing and reasoning context information. Using DRiBAC ontology, the proposed DRiBAC model is able to support semantic and dynamic access control with the following advanced aspects:

- Explicit description of the DRiBAC model
- Dynamic policy enforcement on dynamic interactions among agents in MASs
- Comprehensive and semantic expression of the security relevant context information and its reasoning through the context ontology
- Fine-grained and dynamic access control by incorporating access control and context ontology

7. CONCLUSION

The ability of agents to effectively interact with each other has been the key factor in successful development of open distributed multi-agent systems. However, unauthorized interaction among agents can result in serious security problems. To address this requirement, RiBAC was proposed previously but it provides neither the fine-grained access control for individual interactions among agents nor the ability to capture the context-aware dynamic behaviour of agents.

To overcome the drawbacks of RiBAC, in this paper, we proposed DRiBAC as an extension of RiBAC with new concepts of Role Interaction (RI) and Interaction Assignment (IA), along with four types of context-aware constraints: assignment constraint, activation constraint, cardinality constraint, and SoD constraint. Using RI and IA, DRiBAC provides authorization for actual partnering agents only. In addition, it allows dynamic assignment of roles and permissions and their
adjustment depending on the change in context using context-aware constraints. We defined ontology to represent the DRiBAC model and the relevant context in a well-formed manner and use semantic reasoning about context. To verify the proposed model and ontologies, we presented an implemented prototype based on an online tutoring system. Although we develop DRiBAC ontology in this paper, it is required to develop diverse domain-specific context ontologies for practical verification of DRiBAC.

Although DRiBAC provides a way to control fine-grained interactions among agents dynamically, there exist some limitations as follows:

- Restricted interaction type – DRiBAC considers only one-to-one basis interactions and does not provide a way to deal with different type of interactions such as one-to-many interactions such as auction or many-to-many interactions such as group collaboration. Therefore, the proposed model is suitable for MASs that mainly have one-to-one interactions among agents.
- Less reliable enforcement – In DRiBAC, each agent takes charge of enforcement of access control policies on own resource and tasks. However, agents may enforce policies on the contrary to a system’s decision. DRiBAC sets a high value on agent’s autonomy and advantages of distributed administration than reliable enforcement.

To guarantee more reliable and dynamic interactions in a wide variety of applications, a few things still remain to be investigated in future:

- Model Improvement – To support different types of interactions, we plan to improve DRiBAC with group concern.
- Development of an agent-mediated system – In contrast with the proposed model, in the agent-mediated systems such as AMELI (Esteva, Rosell, Rodriguez-Aguilar, and Arcos, 2004), agents do not interact directly and a mediator takes charge of the institutional enforcement, since those systems put emphasis on centralized but reliable enforcement and administration. To guarantee secure interactions among agents in the e-commerce applications that require highly reliable and available services, we plan to develop a mediated system.
- Development of DRiBAC-based Model Driven Security Model for MASs – Model Driven Security (MDS) is a promising approach to building secure systems (Basin, Doser, and Lodderstedt, 2006). Beydoun et al (2009) propose a metamodel to develop MASs by adopting MDS. Both existing works are based on RBAC. For example, MDS uses SecureUML that based on an extension of RBAC as a security modeling language. If we develop a DRiBAC-based security modeling language for MDS, developers are able to implement MASs which require more fine-grained and dynamic access control in interactions among agents based on MDS. It is worth researching further in future for convenient and rapid development of such interaction-intensive MASs.

REFERENCES
Ontology-based Dynamic Role Interaction Control in Multi-Agent Systems


MINSOO KIM received the Master’s and PhD degree from Ajou University where his research activities involved the developing technique for situation-aware computing. His research interests include context-awareness, access control, semantic web, and multi-agent systems. He is a visiting researcher in the Laboratory of Education and Research on Security Assured Information Systems (LERSAIS) at the University of Pittsburgh and now developing a situation-based access control model and security systems.


BIOGRAPHICAL NOTES

Youna Jung is a research scientist in the Advanced Computing and Information Systems (ACIS) Laboratory at the University of Florida. She received her MS and PhD degrees in computer engineering from Ajou University in South Korea in 2002 and 2007. She worked as a postdoctoral researcher in the Laboratory of Education and Research on Security Assured Information Systems (LERSAIS) at the University of Pittsburgh from 2008 to 2011. Her research interests include situation-aware computing, cooperative computing, community computing, security on intelligent systems, and security on cooperative systems.

Minsoo Kim received the Master’s and PhD degree from Ajou University where his research activities involved the developing technique for situation-aware computing. His research interests include context-awareness, access control, semantic web, and multi-agent systems. He is a visiting researcher in the Laboratory of Education and Research on Security Assured Information Systems (LERSAIS) at the University of Pittsburgh and now developing a situation-based access control model and security systems.