A Flexible Role-Based Delegation Model with Dynamic Delegation Role Structure

Zidong Liu[1], Weiqing Sun[2], and Mansoor Alam[1]
[1]Department of Electrical Engineering and Computer Science, University of Toledo, Toledo, Ohio, USA
[2]Department of Engineering Technology, University of Toledo, Toledo, Ohio, USA

Abstract - *As information systems became widely used by organizations and enterprises, resource sharing and collaboration of work have been pervasive. As a natural way to realize this, delegation has become the routine rather than the exception. However, traditional delegation models have encountered various issues in meeting the growing and diverse requirements. Some of them fail to provide sufficient delegation functionalities, while others are cumbersome to apply and manage practically. Therefore it is imperative to have a flexible delegation model that provides fine-grained control according to different scenarios. Meanwhile, such a model should be easy to apply and maintain. Therefore, we develop a flexible delegation model based on role-based access control which supports fine-grained delegation control at both role and permission levels. Moreover, the proposed model introduces a dynamic delegation role structure to deal with different types of delegation requests. Finally, a prototype was implemented to demonstrate the feasibility of the model.*

Keywords: Role-based access control, Delegation, Delegation model, Delegation role structure

1 Introduction

Nowadays, Role-based access control (RBAC) has become dominant in the access control domain and has been widely used by the majority of organizations, especially large enterprises, hospitals, and colleges due to its simplicity in the management of access rights [1]. In RBAC, individual users are grouped into roles that relate to their positions within an organization and assigned permissions to various roles according to their statures in the organization [2].

The basic idea of delegation is that some active entity in a system delegates authority to another active entity to carry out some functions on behalf of the former [3]. Delegation may occur in two forms: administrative delegation and user delegation. In administrative delegation, even though the administrative user does not possess certain rights, he/she can assign access rights to a user [4]. However, in user delegation, a user delegates the role or a subset of the role to another user to carry out a bunch of functions [4]. Rather than normal access right administration operations, which are performed centrally, delegation operations are usually performed in a distributed manner [5].

Due to the rapid growth of electronic commerce, the online information system has become the mainstream for large organizations. And all the resources required to carry out certain operations are barely local to the system where the user is logged in. In such cases, information sharing tends to be very dynamic and often ad hoc, and delegation is more often the rule than the exception [6]. Under these circumstances, the ideal delegation model should be flexible to cope with different delegation scenarios and reduce administrative costs. However, traditional delegation models have encountered various issues in meeting the diverse and growing delegation requirements. For instance, some existing approaches are focused on the theoretical models without the consideration of practical implementations and management costs.

This paper addresses the delegation issues surrounding user-to-user delegation. We introduce a flexible role-based delegation model. The objective of our model is to fulfill most delegation tasks in a flexible and cost-effective manner. In particular, our model introduces a delegation level decision making function so that delegations could be carried out at both role and permission levels. In addition, a dynamic delegation role structure is integrated into the model to meet the ever changing delegation requirements.

The rest of the paper is organized as follows. We present the related work in Section 2. Section 3 defines and explains the model in detail. Section 4 describes the implementation and evaluation of the model in a healthcare system. Conclusions and future work are presented in Section 5.

2 Related work

There has been much research work to address the delegation issues. The most famous one is RBDM0 (Role-Based Delegation Model 0), which is based on NIST's RBAC model. RBDM0 is the first attempt to model delegation of

authorities which realizes a simple user-to-user delegation of roles. In particular, it formalizes the delegation model with total delegation, which means that each user in a delegation role delegates the total package of permissions embodied in that role. Meanwhile, it defines a can-delegate relationship to control the user-to-user delegation. It also deals with other delegation issues including revocation and multi-step delegation [3], [7].

RBDM1, the successor of RBDM0, adopts the formalization in RBDM0 and extends it to support hierarchical roles. Since the new model was introduced to support hierarchical roles, it also defines different semantics that impact the can-delegate relation [8].

RDM2000 (Role-Based Delegation Model 2000), an extension of RBDM0, was proposed to support delegation in the role hierarchy and multi-step delegation. It develops a rule-based declarative language to specify delegation policies and takes a different approach from RBDM0 to solve the delegation issues [9], [10]. But when a delegator wants to delegate a piece of role, none of the models above can provide a satisfactory solution since the unit of delegation in them is "role", and in many cases, it is necessary and useful to delegate only a subset of the permissions from a role.

While all the models mentioned above were focused only on delegation of roles, PBDM (Permission-Based Delegation Model) was developed to realize delegation based on permissions. It supports partial delegation by separating role sets, and a subset of permissions from a regular role is allowed to be delegated and a new delegation role is created with the set of permissions. PBDM is actually a family of models which extends RDM2000 to incorporate more features. It provides great flexibility in authority management [10], [11]. Although PBDM is a complete model, the controlled propagation on resources is not supported.

The RBDM model which uses sub-role hierarchies supports a variety of delegations. For example, administrators can easily control the permission inheritance behavior by using the restricted inheritance functionality. Roles are divided into a number of sub-roles based on the characteristics of job functions and the degree of inheritance [12]. However, like any other delegation model which is based on the role level, this model does not support permission level delegation.

Another extended RBDM model uses the characteristics of PBDM and the concept of sub-role hierarchies [13]. The advantages of both RBDM and PBDM models are thereby also available in this model. However, the role set in the model is divided into seven layers which add complexity to the realization.

With the development of information technology, traditional delegation models could not keep up with the needs of the ever-evolving information systems. It is a complex task to manage delegation and describe all the delegation requirements in a comprehensive model. Thus, delegation models themselves are extended to support new delegation characteristics [14]. And more and more delegation models were proposed by also taking account of specific delegation needs or organizational structures.

Based on the organizational hierarchies, the organizational supervised delegation model (OSDM) was proposed to identify users who must approve the delegation. The model targets at solving the problem in managing the complexity of the huge number of relations in traditional delegation models [15]. Event-Based Task Delegation aims to reason about the delegation events to specify the delegation policies dynamically to control and secure the delegation process. The model identifies two important issues for delegation, i.e., allowing delegation tasks to carry out, and having a secure delegation within a workflow [16]. RBAC with delegation in a workflow context (DW-RBAC) was proposed to address delegation and revocation in the workflow based systems because these systems have been criticized as being inflexible for the lack of support for delegation [17].

Administration cost is also an important factor in the delegation process; however, many delegation models are cumbersome and hard to apply in real world scenarios though incorporating various features.

In order to provide flexibility and reduce administration costs, the capability-based delegation model was proposed to achieve a higher level of collaboration in large-scale information systems. The approach to model delegation is to integrate a capability-based access control mechanism and map it to permissions as well as roles in each domain, and by means of the assignment of roles to capabilities, suitable permissions are automatically assigned to users [18].

3 A flexible role based delegation model

3.1 Dynamic delegation role structure

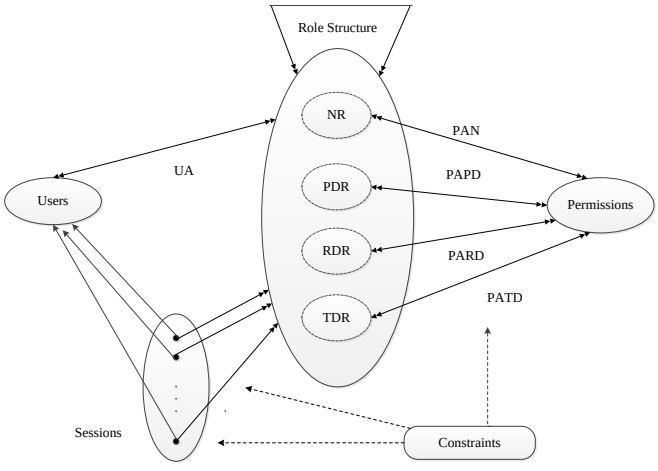

Fig. 1. A flexible delegation model with role structure.

A role in RBAC is the aggregate of responsibility and authority, to which the access to the object is permitted [19]. It is an intuitive way to realize delegation based on the existing role structure. A flexible delegation scheme is desired in many cases as we have mentioned above. Figure 1 shows our proposed model with the structured role sets. In the model, there are four layers of roles: normal roles (NR), predefined delegation roles (PDR), retained delegation roles (RDR) and temporary delegation roles (TDR). A role can be placed in one of the four layers by considering the different delegation scenarios. This leads to a partition of permission-role assignment (PA): permission-normal role assignment (PAN), permission-predefined delegation role assignment (PAPD), permission-retained delegation role assignment (PARD) and permission-temporary delegation role assignment (PATD).

NR is the set of normal roles which can also be used in delegation. PDR, RDR and TDR are the set of roles which can only be used in delegation.

PDR is defined in advance in order to fulfill the common delegation requests in the system. In other words, PDR contributes to the most common delegation scenarios. For instance, in distributed-computing environments, applications or users have to share resources and communicate with each other in order to get their jobs done. And it would be useful to support the delegation scenarios in which a user assigns a subset of the available access rights to another user. However, it is time-consuming to set up the temporary delegation roles for each such request. Therefore, it is more efficient to establish a set of predefined delegation roles for the system. Furthermore, in traditional information system, e.g., healthcare information system, inexperienced personnel such as intern doctors would be delegated typical tasks to be trained to be qualified for the job. A set of PDR would come in handy to fulfill the requirement.

Information is typically classified according to their security characteristics. Likewise, as shown in Table 1, we categorize permissions into three categories based on the information they have access to.

Table 1. Categorization of permissions and characteristics

Degree	**Characteristics**
Unconditional	Permissions which have access to confidential data
Conditional	Permissions which have access to secret data
Restricted	Permissions which have access to top secret data

Normally, a PDR is the sub-role of an NR based on the job functions. A PDR contains conditional permissions. With that, a PDR can be readily assigned to any member in a certain department.

However, the number of PDR should be limited. First of all, the number of conditional permissions of a certain role is limited. Second, in RBAC, a senior role inherits all the permissions from all its junior roles. If a PDR is derived from a senior role, chances are that the PDR is identical with one of its junior roles. In case of that, there is no need to set up a PDR role. Table 2 shows a possible scenario where a PDR and a junior role Assistant Doctor own the same permissions if the PDR is set up improperly in a healthcare information system. In particular, the permissions include create/view electronic patient records (EPR) and view prescription files (PR).

Table 2. Permissions of physician, assistant doctor and PDR

Roles / **Permissions**	**Physician**	**Assistant Doctor**	**PDR**
Create EPR	○	○	○
View EPR	○	○	○
Edit EPR	○	×	×
Delete EPR	○	×	×
View PF	○	○	○

Under this circumstance, Physician inheritances all the permissions of Assistant Doctor, PDR might be identical to role Assistant Doctor if the PDR is set up inappropriately.

Figure 2 illustrates a possible simple role hierarchy and their relationships. Vertically, the senior role holds all the permissions of the junior role, while horizontally, PDR 1 is the sub-role of the Senior role and PDR 2 is the sub-role of Junior role. It is possible that PDR 1 and the Junior role have the same permissions.

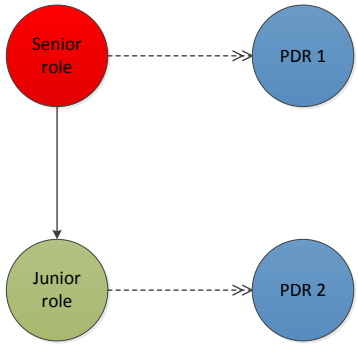

Fig. 2. Example of PDR roles in the role hierarchy.

For infrequent delegation requests, the TDR will be established. In our model, all the TDR roles will be stored in a dynamic buffer. It is used to deal with short-term delegation requests by realizing permission level delegations. When there is no suitable role to fulfill the delegation request, a TDR will be created using the PATD function. A temporary delegation role would be transformed into a retained delegation role (RDR) automatically once it has met certain usage rate and been proved secure under the surveillance of the delegation monitor. Otherwise it will be deleted after certain time period due to the lack of usage. RDR roles will be readily used to

serve repeated delegation requests and therefore can reduce runtime overhead to create new TDR roles.

3.2 Delegation decision function

As discussed in Section 3.1, our proposed model provides four different delegation layers to cope with different delegation scenarios. However, there is a need to provide a systematic method to identify and select an appropriate layer from the role structure to fulfill the delegation requests. Therefore, we introduce the *Role_of_Delegation* function, which is in charge of selecting the appropriate delegation levels. It describes how the decision for the delegation level is made according to different delegation requests by taking users, operations and objects as input, and querying the current delegation role structure.

Role_of_Delegation: SESSIONS×OPS×PERS×OBS→ROLE or NULL

- ROLE: There is a suitable delegation role in the role structure for the delegation, and then the delegation will be executed at the role level.
- NULL: There are no suitable roles in the role structure, and in this case, the delegation has to be performed at the permission level, which means a partial delegation from a single or multiple roles is needed.

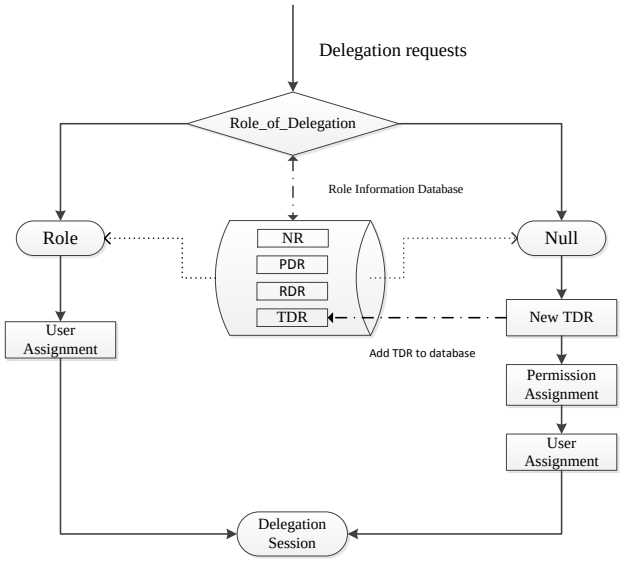

Fig. 3. Process flow-chart of the model.

Figure 3 shows the logical flow-chart for the model. The Role_of_Delegation function interacts with the role structure and checks if there is a suitable role for the request. If there exists such a role, the function will simply return it. Otherwise, the function returns NULL, and then a new TDR will be created and assigned with the permissions specified in the request. By this way, both role level and permission level delegations can be realized.

3.3 Multi-step delegation

Multi-step delegation is also an important delegation feature as it allows the delegated role memberships to be further delegated to other roles. Delegation depth denotes how many times the task has been delegated in a delegation chain. In multi-step delegation, the depth of the delegation chain is greater than one, and delegated permissions can be further delegated multiple times. Multi-step delegation is useful in a variety of organizations. For example, in healthcare information systems, unskilled delegatees like intern doctors or assistant doctors might not be able to handle the task independently, so the delegation has to be further delegated.

In traditional delegation models, permissions assigned to the delegation role are constant in the delegation chain. For example, Alice assigns a delegation role which could be her role or sub-role to Bob, and Bob further delegates the same role to Cathy. Although this arrangement simplifies the management, it is not flexible to handle certain delegation requests. For instance, a physician pays a visit to another hospital and delegates his/her role to the assistant. Later on, the assistant finds himself overloaded with other work and cannot fulfill the all the tasks alone, so he further delegates the role to an intern doctor who has time and skills to complete the task. However, due to the security policies, an intern doctor could not possess some of permissions. Usually, such a delegation request will be rejected and the assistant has to find another user to further delegate the entire privileges. However, it is possible that the delegation request be rejected again. In emergency situations, it will bring obstacles in carrying out the delegation in a timely manner.

In our model, the delegation chain is more flexible. The delegator may modify the delegation role to further assign it to the next delegatee. However, the subsequent delegator can only remove some of the original delegation privileges and further assign it to other users. For instance, Alice delegates her role to Bob; therefore a delegation chain is created. Later on, Bob further delegates the role to Eric; he can only re-delegate the role or a part of role. Such a scheme would be able to address the problem raised in the previous example, and in that case, the assistant is able to only delegate some of the physician's permissions to the intern and keep the rest; and the whole operation can be performed in a timely manner. Moreover, such a scheme is easy to implement in our model. Once the original delegation role is modified in the delegation chain, the proposed role structure and decision function would collaborate to make the appropriate delegation choice.

3.4 Delegation revocation

Since the proposed model supports delegation at both role level and permission level, multiple revocation schemes are supported to make the model robust and flexible, including timeouts and requiring the original member of the delegating role to revoke the delegation. Meanwhile, it is also

possible to execute the revocation by revoking a user from the delegation role or removing a subset of permissions from that delegation role.

Furthermore, due to the structural delegation role sets, revocation schemes are also layered. Simple revocation schemes lay at the bottom level, while fine-grained control revocation schemes sit at the top. For instance, when a delegation role PDR is performing, it corresponds to a relaxed revocation scheme because it is considered as relatively secure. In this case, multiple revocation schemes are supported, such as grant-dependent revocation in which only the delegator is allowed to revoke the delegated privileges from the delegatees. In the meantime, noncascading revocation is supported which corresponds to multi-step delegation, in which the delegation between the second and the third user will not be revoked even when the first user revokes the delegation. For instance, in the multi-step delegation scenario, a PDR is delegated from Alice to Bob, and eventually transferred to Eric, the noncascading revocation is allowed in such a case. The revocation steps would be as follows.

1) Select the PDR which is delegated by Alice.
2) Remove the PDR in the first step of the delegation, in order to revoke the roles assigned to Bob.
3) The delegation chain from Alice to Bob is revoked.
4) Select the PDR which is delegated by Bob.
5) Remove PDR in the second step of the delegation, in order to revoke the roles assigned to Eric.
6) The delegation chain from Bob to Eric is revoked.
7) The entire delegation chain is revoked.

Even though Alice and Bob use the same PDR, the delegation chain has to be revoked step by step.

Noncascading and cascading revocation are both allowed if a PDR is used in the delegation. However, when the delegation request is from outside of an organization and a TDR is used, it would be considered less secure and the delegation should be under higher security supervision and demand a stricter revocation. Normally, once there are some abnormalities or when violations are detected, any user who has the privileges can revoke the delegation and the entire delegation role would be revoked. In this situation, if the delegation between the first user and the second user is revoked, the entire delegation chain will be revoked. Noncascading revocation is not allowed in delegations with a TDR. The revocation steps would be as follows.

1) Select the new created TDR which is delegated by the original delegator.
2) Remove the permissions associated with the TDR, which means to revoke the PATD function.
3) Remove TDR so as to revoke the roles assigned.
4) The entire delegation chain is revoked.

4 Implementation and evaluation

4.1 Implementation of the model

A prototype has been developed to verify the feasibility of the proposed model. It is based on the open source software OpenEMR (Open Electronic Medical Record) [20]. We chose it as our platform because it provides features such as electronic health records management, practice management, scheduling, and electronic billing. Meanwhile, it provides a set of predefined roles such as clinicians and physicians. However, OpenEMR does not have a delegation mechanism in its current version.

We designed the delegation server which handles the delegation requests and controls the delegation process based on the proposed model. We use PHP for the implementation of the prototype, and use MySQL to store all the delegation related data. OpenEMR uses phpGACL (Generic Access Control Lists) module to realize role-based access control [21]. And our delegation server makes use of functions provided by phpGACL to instantiate the delegation role structure. Delegation requests are submitted through the delegation request form as shown in Figure 4.

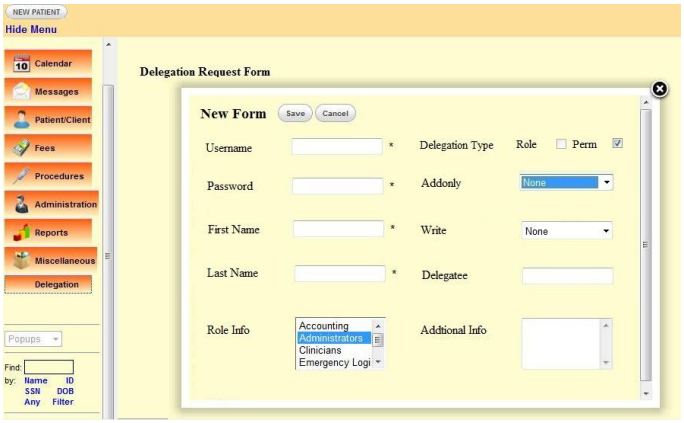

Fig. 4. Delegation request from.

In the delegation request form, the first four fields are used for authentication. “Role Info” is the role that the delegator currently holds. The delegator needs to specify whether he/she wants to delegate the entire role or just some of the permissions in the delegation type option. If the role delegation type is chosen, then the entire role will be delegated. If permission delegation type is selected, the delegator needs to specify the permissions and objects he/she wants to delegate. “Addonly” and “Write” are the two fields which store detailed operations associated with the objects. Upon the completion of the form, the delegation request will be sent to the delegation server. Then the delegation server will search the current delegation role structure for suitable existing roles for delegation. A TDR will be automatically created if no such role is identified.

The delegator can customize the permissions by manipulating the "Permissions" and "Objects" fields in the request form. In addition, the delegatee can be selected. Once the delegator submits the form, the delegation request will be sent to the delegation server for further processing.

Delegation server is the core component in the implementation. On the one hand, it accepts delegation requests, conducts checks and controls the execution of the delegation procedure based on the result of *Role_of_Delegation* function; on the other hand, it receives the feedbacks from Delegation monitor at the end of the delegation session, which includes useful information like whether the delegation is successful or not, what kinds of errors in case of failure, and so on. The feedbacks will be used to optimize the selection of delegation levels, delegation roles and delegatees. Each time the delegation task is fulfilled, the information about the delegation roles used in the process would be automatically stored or updated in the role set database.

Furthermore, the administrator monitors and manages all the delegation logs. All the denied delegation requests and all the unfulfilled delegation requests will be identified. By doing so, one can analyze the failed delegation or further realize a delegation scoring mechanism to select the best delegatee.

4.2 Evaluation of the model

In this Section, we simulate different scenarios in user-to-user delegation based on our prototype system to evaluate the functionality and performance of the model. Initially we set up a predefined delegation role Physician_intern which would be used when a physician delegated some permission to an intern doctor. The Physician_intern role consists of permissions (Documents (read), Medical history (read/write)).

Suppose that Alice is a physician in the hospital. And she wanted to delegate some permission to an intern doctor Bob by sending a delegation request which contained the permissions (Documents (read) and Medical history (read/write)). The authorization check will pass as there are no security violations. After that, the *Role_of_Delegation* function returned Physician_intern, which is exactly the desirable delegation role. Then the delegation session started. We name this delegation operation DO1.

Likewise, we changed the delegation role set by adding two more predefined delegation roles including Surgeon_assit and Surgeon_intern.

Then another physician Bill wanted to delegate a part of his permissions (Documents (read/write) and Medical history (read/write)) to the assistant Dan, he simply sent a delegation request message to the delegation server. Then the *Role_of_Delegation* function returned NULL, which meant there is no feasible delegation role under current circumstances. In this case, the system would create a temporary delegation role in which permissions the physician referred to were incorporated to perform the delegation task. When the physician's assistant logged in, the delegation module would notify Dan that the delegation role with the necessary permissions have been assigned in order to fulfill the delegation task. When the delegation session was expired, the delegation monitor would check if the task was fulfilled successfully. After that, the monitor would send a feedback to the delegation server and the temporary delegation role TDR1 would be saved in the temporary delegation role buffer. We call this delegation operation DO2.

Later, the same delegation request was issued by Alice to delegate the same set of permissions to her assistant, this time the *Role_of_Delegation* function returned TDR1, which indicated that the saved temporary delegation role created earlier in the role structure would work directly. In the prototype implementation, there is a delegation frequency for each delegation role. A temporary delegation role would become a retained delegation role once the frequency reaches 10. In the evaluation, we used the TDR1 for sufficient times so that it turned into a retained delegation role (RDR1). In this case, *Role_of_Delegation* returned RDR1 which was deemed to be more trustworthy. This time, the delegation operation is referred to as DO3. Table 3 shows the changes of the role structure between each of the delegation operations.

Table 3. Performance evaluation

Delegation Operations	Number of Roles				Request Fulfilling Time	
	NR	PDR	RDR	TDR	PBDM	Our Model
DO1	6	1	0	0	0.52s	0.56s
DO2	6	3	0	1	0.55s	0.38s
DO3	6	3	1	0	0.54s	0.39s

Initially, there are only 6 normal roles which were set up in advance, so the number of NR is constant. The numbers of RDR and TDR are dynamically changing due to the change of the delegation frequency. This evaluation showed the flexibility of our model because it can make appropriate delegation choices according to different delegation requirements. Meanwhile, the retained delegation role set in our model is dynamic as one temporary delegation role could be transformed into a retained delegation role while another could be removed from the temporary delegation role structure. By doing so, the model would be adaptive to the ongoing delegation requests.

Then we evaluated the multi-step delegation. This time Bill delegated the same task to his assistant Dan. However, Dan was not able to finish the task and he further delegated the task to Bob. In both steps of the delegation, *Role_of_Delegation* returned RDR1 because there was no modification in the delegation requests. After the user

assignment indicated that the delegation would be performed at role level, the delegation session started.

In addition, we did a performance comparison between our model and a simple PBDM model implementation based on the prototype system. In particular, we implemented PBDM by disabling *Role_of_Delegation* so that delegation could only be performed at the permission level. In the evaluation, we gradually increased the number of roles to compare the efficiency between the two models. We monitored the simple delegation scenarios exactly as mentioned above and recorded the request fulfilling time which is the total elapsed time to fulfill the delegation request.

The results show that the request fulfilling time for PBDM is almost constant because each time a delegation contains a piece of permissions, a new temporary delegation role would be created, and the time is not affected by the number of roles in the role structure. Our model was slower than PBDM initially, because the number of roles was limited and most of the delegations had to be performed at the permission level, and our model had to call the *Role_of_Delegation* function for the delegation decision making. As the number of roles increased, the request fulfilling time of our model decreased because more and more delegation requests were being performed at the role level. The evaluation showed that our model is efficient and adaptive in typical delegation scenarios in a healthcare information system, and the model can help to accelerate the delegation decision making process by identifying the suitable delegation levels and roles automatically.

5 Conclusions and future work

This paper proposes a flexible delegation model in an effort to fulfill a variety of delegation scenarios. The dynamic role structure, as the key component in the model, enables delegation requests to be fulfilled at suitable levels automatically. The model has been shown to be effective and efficient through a prototype implementation and evaluation in a healthcare information system. However, there is still room for further improvement of our model. Multiple delegation and role-to-role delegation will be incorporated in order to support a wider range of delegation scenarios.

6 References

[1] D. F. Ferraiiolo, R. Sandhu, S. Gavrila, D. R. Kuhn and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Transactions. On Information and System Security*, vol. 4, pp.224-274, Aug. 2001.

[2] J. Wainer and A. Kumar, "A fine-grained, controllable, user-to-user delegation method in RBAC," *Proc. The 10th Symposium on Access Control Models and Technologies*, New York, NY, June. 2005, pp 59-66.

[3] Ezedin Barka and Ravi Sandhu, "A Role-Based Delegation Model and Some Extensions," *Proc. The 23rd National Information Systems Security Conference,"* Dec. 2000.

[4] J. Crampton and H. Khambhammettu, "Delegation in role-based access control," *Proc. The 11th European Symposium on Research in Computer Security*, Berlin, Heidelberg, 2006, pp. 174-191.

[5] Qihua Wang, Ninghui Li and Hong Che, "On the Security of Delegation in Access Control Systems," Proc. The 13th European Symposium on Research in Computer Security, Berlin, Heidelberg, 2008, pp. 317-332.

[6] Longhua Zhang, Gail-Joon Ahn and bei-Tseng Chu, "A rule-based delegation framework for healthcare information systems," Proc. The 7th ACM symposium on Access control models and technologies, New York, NY, June. 2002, pp. 125-134.

[7] Ezedin Barka and Ravi Sandhu, "Framework for Role-Based Delegation Models," Proc. The 16th Annual Computer Security Applications Conference, Washington, DC, Dec. 2000.

[8] Ezedin Barka and Ravi Sandhu, "Role-Based Delegation Model/ Hierarchical Roles (RBDM1)," Proc. The 20th Annual Computer Security Applications Conference, Washington, DC, Dec. 2004, pp. 396-404.

[9] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu, "A rule-based Framework for Role-Based Delegation," Proc. The 6th ACM Symposium on Access Control Models and Technologies, New York, NY, May 2001, pp. 153-162.

[10] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu, "A rule-based Framework for Role-Based Delegation and revocation," ACM Transactions on Information and System Security (TISSEC), vol. 6, pp. 404-441, Aug. 2003.

[11] X. Zhang, S. Oh, and R. Sandhu, "PBDM: a flexible delegation model in RBAC," Proc. The 8th ACM symposium on Access control models and technologies, New York, NY, June 2003, pp. 149-157.

[12] HyungHyo Lee, YoungRok Lee, and BongNam Noh, "A new role-based delegation model using sub-role hierarchies," P*roc. The 18th International Symposium on Computer and Information Sciences (LSCIS 2003)*, Antalya, Turkey, Nov. 2003, pp. 811-818.

[13] Dong-Gue Park and You-Ri Lee, "A flexible role-based delegation model using characteristics of permissions," *Proc. The 16th international conference on Database and Expert Systems Applications*, Berlin, Heidelberg, 2005, pp. 310-323.

[14] Meriam Ben-Ghorbel-Talbi, Frédéric Cuppens, Nora Cuppens-Boulahia, and Adel Bouhoula, "A delegation model for extended RBAC," *International Journal of Information Security*, vol. 9, pp. 209-236, June 2010.

[15] Nezar Nassar, Nidal Aboudagga, and Eric Steegmans, "An Organizational Supervised Delegation Model for RBAC," *Proc. The 15th international conference on Information Security*, Berlin, Heidelberg, Sept. 2012, pp. 322-337.

[16] Khaled Gaaloul, Ehtesham Zahoor, François Charoy, and Claude Godart, "Dynamic Authorisation Policies for Event-based Task Delegation," *Proc. The 22nd International Conference on Advanced Information Systems Engineering*, Berlin, Heidelberg, June. 2010, pp. 135-149.

[17] Jacques Wainer, Akhil Kumar, and Paulo Barthelmess, "A formal security model of delegation and revocation in workflow systems," Information Systems - IS, Vol. 32, pp. 365-384, May 2007.

[18] Koji Hasebe, Mitsuhiro Mabuchi, and Akira Matsushita, "Capability-based delegation model in RBAC," *Proc. The 15th ACM symposium on Access control models and technologies*, New York, NY, June. 2010, pp. 109-118.

[19] E.C. Lupu, D. A. Marriott, M. S. Sloman, and N. Yialelis, "A Policy Based Framework for Access Control," *Proc. The First ACM Workshop on Role-based Access Control",* New York, NY, Dec. 1996.

[20] E. Helms and L. Williams, "Evaluating Access of Open Source Electronic Health Record Systems." *Proc. 3th International Conference on Software Engineering*, New York, NY, May. 2011, pp. 73-80.

[21] Mike Benoit, James Russell, and Karsten Dambekalns, *Generic Access Control Lists with PHP*, Sept. 2006.