2011 Workshops of International Conference on Advanced Information Networking and Applications

A Hybrid Sharing Control Model for Context Sharing and Privacy in Collaborative Systems

Ahmad Kamran Malik, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology, Austria
{kamran, dustdar}@infosys.tuwien.ac.at

Abstract – **Complex Web-based information systems involving multiple entities and their dynamic mobile-based collaborations require efficient techniques for context information sharing. Sharing control is a requirement for preserving the privacy of personal context and shared context. Our sharing control mechanism is hybrid, based on sharing control rules defined by enterprise as well as by individuals users. Our complex scenario involves multiple entities which require prioritization and conflict handling mechanism for entities and their policy rules. This paper presents a sharing control model, Web services-based architecture and its implementation with a running example. The system is evaluated by comparing our hybrid sharing control policy with enterprise-defined role -based policy and shows effectiveness of our hybrid policy in collaborative information sharing environments.**

Keywords-Context Sharing, collaborative system, hybrid sharing policy, access control, conflict handling.

I. INTRODUCTION

Creating sharing control policy for an information sharing system with multiple entities involved with their conflicting interests is a complex task [1] that requires management of information at different granularity levels and complex rules for information sharing and conflict handling strategies. In current Web-based collaborative systems like [2], dynamic involvement of enterprises, teams, and individual users is increasing day by day. Dynamic teams are created by collaborating enterprises, distributed and mobile-based users are involved in them; a user can join or leave a team at any time and a user can be a part of more than one teams at a time. Collaborative systems must be flexible enough to handle sharing requirements of dynamic entities, but not at the cost of privacy of users, their teams, and enterprises. Role-based Access Control (RBAC) policies [3] cannot handle fine grained access requirements at the level of individual user, activity, team or enterprise with multiple dynamic entities. We use a hybrid policy model including enterprise-defined policy and user-defined policy and extend role-based policy by adding context constraints and user-defined rules. Our user-defined rule-based policy overrides enterprise-defined policy for privacy of user's personal services and shared services related to her team activities. Conflicts among sharing control policy rules and entity priorities needs to be handled for an efficient policy evaluation [4].

In this paper we describe a sharing control model based on users collaborations involving multiple entities in Collaborative Working Environment (CWE). From this model we use entity priorities and collaborative relationships to define sharing control rule, rule priorities for all involved entities, and their conflict handling mechanism.

In our system we extend RBAC [3] using context constraints as well as our rule-based user-defined policy. Our sharing control mechanism is based on two types of policy rules including enterprise-defined RBAC rules and user-defined rules. This hybrid policy controls sharing of context at certain level of detail using collaborative relationships and entity priorities. An architecture of the system and Web services-based implementation with a running example describing our hybrid sharing control policy is provided which shows the importance of its applications in the real world scenarios. Evaluation of the system is described by comparing our user-defined hybrid sharing control policy with the enterprise-defined role-based policy. Results describe effectiveness of our hybrid sharing control policy which provides increased number of responses as well as level of responses among the entity types which are close to each other. It is indicator of better preserving user privacy and increased level of sharing among collaborating users who are in close relationships, for example, involved in same activity or team.

The remainder of the paper is organized as follows. Section II describes background and related work. Section III explains sharing control model for collaborative working environments. Section IV describes our sharing control policy. Section V discusses our sharing control architecture and evaluation. Section VI concludes the paper and describes future work.

II. BACKGROUND AND RELATED WORK

Access policy for collaborative environments and their requirements have been described in [5]. Role-Based Access Control model [3] defines roles using permissions to access objects. RBAC is an efficient model for management of rights in large scale systems, it is rather static and lacks in fulfilling requirements of collaborative systems to access objects at level of individual user and other entities in a dynamic environment.

RBAC can be used in collaborative systems as explained in [6] and in a survey by [5]. Context-based RBAC systems have been an interesting area of research as can be seen in [7] and [8]. Web-based systems also make use of RBAC model, for example, system using Web services like [9] and [10]. Our

978-0-7695-4338-3/11 $26.00 © 2011 IEEE
DOI 10.1109/WAINA.2011.134

system also makes use of Web services for sharing context and information among collaborating users. An owner centric access control based on owner created roles is presented in [11]. Another system for sharing context and preserving user's privacy using owner-defined roles is presented in [12].

Policy conflicts in complex systems can be handled by defining priorities for different entities or groups and types of rules, for example, positive and negative rules [4]. Authorization policies, obligation policies, and their conflicts resolving strategies are described in [13]. Examples and details of conflict management strategies for organizational based access control are described in [14]. Static and dynamic conflict are described and their detection methods are provided in [15]. We define priorities and conflict handling rules for these hybrid policies and all entities involved in the system.

III. SHARING CONTROL MODEL FOR COLLABORATIVE WORKING ENVIRONMENTS

In CWE, users need to share their context with collaborating partners for accomplishing shared activities. Owners need to control their data being shared with other users using owner-defined rules while enterprises want to define role-based access control rules for their employees. In this scenario, we use the term sharing control for two purposes - firstly, it describes control of owner on her data being shared, and secondly, it describes sharing of control decisions between owner and enterprise.

A. Entities, Entity Levels, and Priorities

Our system uses five entity types; enterprise, team, activity, role, and user as shown in Figure 1 with enterprise being the lowest priority level and user being the highest priority level.

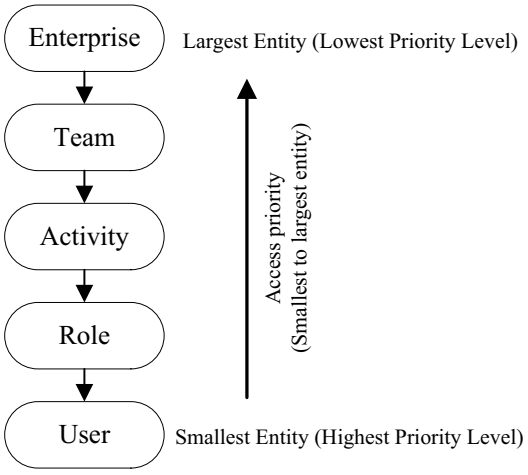

Fig. 1. Entities and their priority levels

Enterprise- It is the largest entity and has lowest priority among entities in our system. Enterprise creates teams, sharing policies, assigns activities and users to team, and roles to users.
Team- Team is the entity that can be created by one or more enterprises. A team is headed by a team leader who assigns users to activities. A user can be part of more than one team at a time.
Activity- Activities can be defined by an enterprise or by team itself. Team leader selects appropriate users for performing activities.
Role- Roles are assigned to users based on their job, qualification, and experience. One or more users can have same role and many roles can be assigned to one user.
User- A user works for an enterprise in different teams and activities. A user can share her context with other collaborating users and can define her own sharing control rules.

B. Collaborative Relationships

Collaborative relationships are the backbone of our system and are described as *Member*, *Mutual*, and *Colleague* as depicted in Figure 2 which shows relationships between users performing activities in overlapping teams belonging to different enterprises.

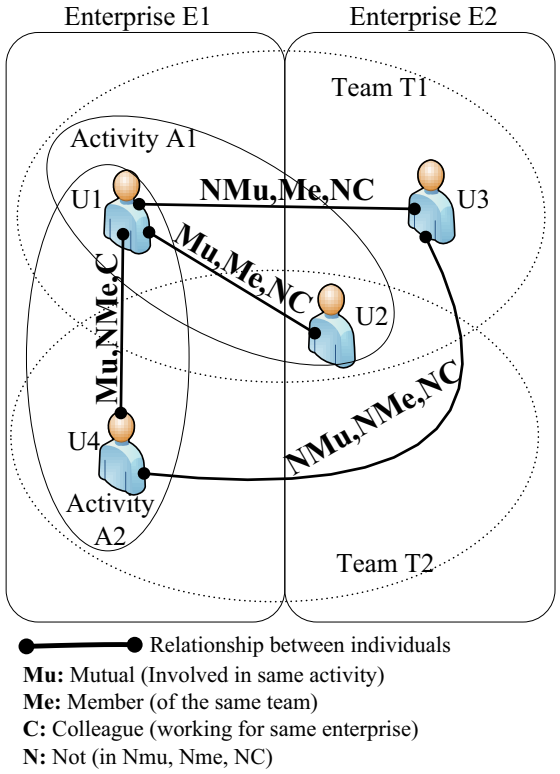

Fig. 2. Types of collaborative relationship

Member relationship is described as *Me* while *non-member* is described as *NMe*. Similarly *Mu* and *C* describe *mutual* and *colleague* while *non-mutual* and *non-colleague* are described as *NMu* and *NC* respectively.
Member(Me)- This relationship describes all those users who are working for same team.
Mutual(Mu)- This relationship between two users describe that both users are part of same activity. Mutual is the closest in all relationships resulting in maximum number of collaborations within and across teams and enterprises.

Colleague(C)- This relationship between two users describes that both users are working for same enterprise. Colleagues are not always part of same team and activities. There can exist complex relationships between collaborating users. For example there can exist a *member* relationship between two users while not being mutual and colleague as shown in Figure 2 between users *U1* and *U3*.

IV. SHARING CONTROL POLICY

Our collaboration scenario involves a number of collaborating entities in complex relationships which requires a dynamic policy to handle these collaborations. We use a hybrid approach including role-based and rule-based sharing control policy. User defined rules are used to protect and share her own information by overriding enterprise defined rules. We still include role-based policies as a part of our system due to fact that they provide better management of large number of users in enterprise, and most of the enterprises are based on them.

A. Hybrid Sharing Control Policy Model

Two types of sharing control policy rules are used in our system; enterprise defined role-based policy and user defined rule-based policy. Enterprise-defined policy is based on role of user which we extend with context conditions. On the other hand user-defined policy is rule-based which contains rule conditions for all entity types and can define positive as well as negative rules to allow and restrict access respectively in certain conditions.

Our rule-based sharing control policy contain two types of rules, unconditional and conditional. Unconditional rules are used in some specific situations, for example, when granting or restricting one or more services unconditionally to an entity like user, team etc.

Authorization Rules

User-defined sharing control policy includes authorization rules that are used to grant access to requesting users. There are two types of Authorization rules which are *RuleType* and *Action*. *RuleType* consists of two values *regular* and *exceptional* whereas *Action* consists of values *allow* and *deny* which are represented by + and - sign respectively.

The basic form of user-defined authorization rule is defined as:

$$< Subject, Object, Condition, AccessLevel >,$$

where

- *subject* is one of the entity types defined in Figure 1. It means authorization rule can allow access to different granularity levels, for example, a single user as well as to a group of users in a team, activity, role, or enterprise.
- *object* is a service that provides information related to a user's personal or shared context.
- *condition* consists of union of clauses where each clause is an intersection of one or more statements. Each statement is described as *(context_attribute operation value)*.
- *access level* describes context in three levels of hierarchy *L1*, *L2*, and *L3* which are used to grant relevant details of context to a requester.

Conditions described in authorization rules must be fulfilled for sharing required services. Conditions are specified in this system as follows.

$$condition := clause \cup clause... \cup clause$$
$$clause := stemenet \cap statement... \cap statement$$
$$statement :=< context >< OP > \{< value > | < context >\}$$

B. Priority and Conflict Handling Policy

Conflicts among rules often occur in rule-based sharing control systems. In presence of many entities and their priority conflicts, a policy is required that can define priorities for entities and their rule conflicts. Group-based and positive/negative rule-based priorities have been used in literature [4].

- **Group-based Priorities:** are used where large number of entities or groups are involved in sharing control similar to our collaborative sharing scenario. Each entity is assigned a priority level which decides sharing conflicts within entities.
- **Positive/Negative Authorization:** Positive rules are used to allow access to a service, while negative rules are used to deny access to service. In some systems where both positive and negative rules are used, positive or negative rules are given priority over the other to handle conflict between rules.

Still there can be some requirements that cannot be handled by group-based and rule-based priorities, these special cases can be handled by defining exceptional priority rules for emergency context [14]. Priority and conflict handling methods are shown in Algorithm 1.

Algorithm 1 Priority and Conflict Handling Algorithm

```
[Find Rules for Policy Evaluation]
if found a rule without conflict then
    Apply negative rule
else if found conflicting rules then
    [Find and Compare Entities in Rules]
    if found exceptional rule then
        Apply this rule
    else if found smaller entity in rule then
        Apply this rule
    else if found same level entity rules then
        Apply negative rule
        Reply "Service Unavailable"
    end if
else if no authorization found then
    Apply default closed policy
end if
```

In addition to priority rules there are situations when no authorization rule is defined for a service, in this case default open or closed rules can be used.

- **Default Open Policy:** Access to service is allowed in case there exist no negative authorization rule.
- **Default Closed Policy:** Access to service is denied in case there exist no positive authorization rule.

Entity-based Priority

These are priority rule that define priority of one entity over another in case of conflict between them. In our system, there exist entities namely user, role, activity, team, and enterprise. Conflicting rules may exist in system depending on a user's involvement in these entities. For example a user *U* is part of team *T* and is involved in activity *A*. Sharing *activity service* of user *U* is allowed to users involved in activity *A* but is not allowed to other members of team *T*. In this case rule will be evaluated based on entity (activity and team) that has higher priority. Thus if activity *A* has higher priority than team *T*, user will be allowed to access required service.

Entities involved in our collaborative scenario can be hierarchically arranged as shown in Figure 1.Smaller entities get higher priority than larger ones because close relationship between users exist in smaller entities. If there is a conflict between positive and negative rules, firstly it is handled using entity priority. When both rules belong to same entity, negative rule is given priority over positive rule and sharing is not allowed.

Exceptional Priority

In situations where entity-based priority rules are not sufficient, exceptional priority rules are only way to fulfill user's sharing requirements [14]. For example, a user wants to restrict whole team *T* to share services *S* belonging to her. Some users of team *T* working in different activities already have access to service *S*. According to given policy smaller entities get higher priority, so the restriction on whole team cannot be fully implemented in presence of small entity priority rules. One solution is to revoke all rights from smaller entities of that team. Obviously, it is very difficult and time consuming to search all grants and revoke them one by one. Additionally, these smaller entity rules may be required after some time whose reassignment will take some time again. Exceptional priority rules solve this problem by defining a rule at any entity level, with or without conditions.

V. SHARING CONTROL ARCHITECTURE AND EVALUATION

In this section, we describe architecture and Java Web services based implementation of our system. We evaluate our sharing control policy with an example explaining different types of sharing control rules, priority conflicts and, a user request evaluation.

A. Sharing Control Architecture and Implementation

Our *Sharing Control architecture* is shown in Figure 3. It is a peer to peer and Web services-based architecture which uses

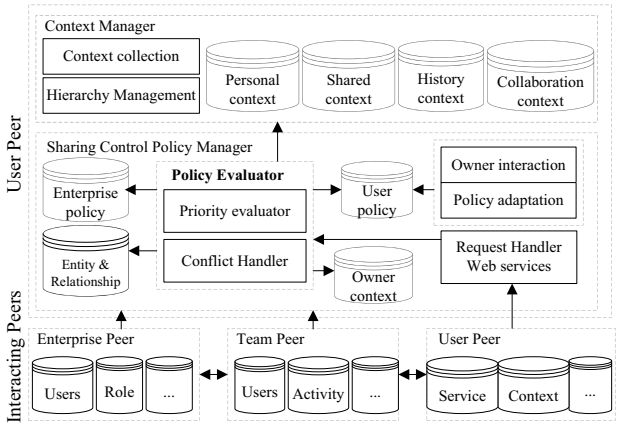

Fig. 3. Sharing Control Architecture

Web services to share context information among collaborating users on the Web. In this system, *sharing control policy manager* contains policy databases and entity relationships. Conflict handler components handles conflict in policy rules detected by policy and priority evaluators. Requested Web services are provided if policy evaluator allows by evaluating all context conditions in relevant rules. User peer gets its role, team, and, activity related data from connecting team and enterprise peers. An individual can query or subscribe for certain type of context with other member of her activity, team, or enterprise.

Implementation of sharing control architecture uses Java Web services and peer to peer technologies. A sharing control messenger application is used to share context among collaborating users. Details of sharing control messenger is out of scope of this paper and is described in DySCon [10].

B. An Example of Sharing Control Policy

Here we describe working of our hybrid sharing control policy with an example. Sharing control rules are explained with example containing regular and exceptional rules as well as positive and negative rules. Context sharing request from a user and its evaluation is also described. User-defined policy is described with examples. This example uses our collaboration scenario shown in Figure 2. Listing 1 contains descriptions for regular rules while Listing 2 describes exceptional rules.

Positive rules allow required service access are shown using symbol "+" whereas negative rules restrict service access are shown using a symbol "-". A regular rule defined by user *U1* of Figure 2 in this example allows members of team *T1* to access *activity service* of activity *A1* as long as the activity is not finished. *Activity service* can be accessed at level *L1* and *L2* being mutual *Mu* and non-mutual *NMu* respectively. Level L1 describes highest granularity level (maximum detail) of context and L3 being the lowest granularity level.

It can be seen from Figure 2 that user *U3* can also access the activity service at level L2 being member of team *T1* and *NMu*. Using the above described rule users who are not part of activity *A1* and are not colleagues of user *U1* can also access *activity A1 service*, for example, user U3. Observing

```
<UserRules>
  <RuleType="regular">
    <Action="+">
      <Subject>
        <Predicate>
          <Op>eq</Op>
          <EntityName> team </EntityName>
          <EntityFunc> name </EntityFunc>
          <EntityValue>T1</EntityValue>
        </Predicate>
      </Subject>
      <Object> activity A1 service </Object>
      <Condition>
        <Predicate>
          <Op>eq</Op>
          <EntityName> activity </EntityName>
          <EntityFunc> name </EntityFunc>
          <EntityValue>A1</EntityValue>
        </Predicate>
        <Exp>AND</Exp>
        <Predicate>
          <Op>neq</Op>
          <EntityName> activity </EntityName>
          <EntityFunc> status </EntityFunc>
          <EntityValue> finished </EntityValue>
        </Predicate>
      </Condition>
      <AccessLevel>
          <Predicate>
            <Op>eq<Op>
            <EntityName> Relationship </EntityName>
            <EntityValue> Mu </EntityValue>
          </Predicate>
          <Level> L1 </Level>
          <Predicate>
            <Op>eq<Op>
            <EntityName> Relationship </EntityName>
            <EntityValue> NMu </EntityValue>
          </Predicate>
          <Level> L2 </Level>
      </AccessLevel>
    </Action>
  </RuleType>
</UserRules>
```

Listing 1. An example of user-defined Rule-based policy by user U1 in Fig. 2

this problem, user *U1* now wants to restrict all those members of Enterprise *E2* from accessing *activity A1 service* who are non mutual *NMu* other than team leaders.

It is not possible to restrict enterprise entity in presence of team entity authorization rule because team entity has higher priority than enterprise entity. In this case an exceptional rule is defined here which shows the use of negative rules as well as power of role tree as shown in Listing 2. Exceptional rules solve most of the conflicts in regular rules. If two exceptional rules or two regular rules have same priority level then negative rules get precedence to keep owner's privacy.

Context information retrieval

A user can send a request to access a context service of other user. Context sharing request is described here which contains subject, object and access level. Suppose that user *U3* sends a context sharing request described in Listing 3 to user *U1* for accessing *activity A1 service*. On receiving this request the requested *Web service* component in Figure 3 sends this request to *policy evaluator* component. *Priority evaluator* component using *entity & relationship* database finds that requester *U3* belongs to Enterprise *E2* and there is an exceptional rule for employees of *E2* in *user policy* database who want to access *activity A1 service*.

Exceptional rule overrides all regular rules for this service,

```
<UserRules>
  <RuleType="exceptional">
    <Action="-">
      <Subject>
        </Predicate>
            <Op>eq<Op>
            <EntityName> Enterprise </EntityName>
            <EntityFunc> name </EntityFunc>
            <EntityValue> E2 </EntityValue>
          </Predicate>
      </Subject>
      <Object> activity A1 service </Object>
      <Condition>
        <Predicate>
          <Op>eq<Op>
          <EntityName> RoleTree <EntityName>
          <EntityFunc> rolename </EntityFunc>
          <EntityValue> Leader </EntityValue>
        </Predicate>
        <Exp>AND</Exp>
        <Predicate>
          <Op>eq<Op>
          <EntityName> RoleTree <EntityName>
          <EntityFunc> hierarchy </EntityFunc>
          <EntityValue> down </EntityValue>
        </Predicate>
        <Exp>AND</Exp>
        </Predicate>
            <Op>eq<Op>
            <EntityName> Relationship </EntityName>
            <EntityValue> NMu </EntityValue>
          </Predicate>
      </Condition>
    </Action>
  </RuleType>
</UserRules>
```

Listing 2. An example of user-defined Exceptional Priority Rule

```
<SharingRequest>
  <Subject>
    <EntityName> User </EntityName>
    <EntityFunc> name </EntityFunc>
    <EntityValue> U3 </EntityValue>
  </Subject>
  <Object> activity A1 service </Object>
  <credentials>...</credentials>
</SharingRequest>
```

Listing 3. Context sharing request

so *policy evaluator* evaluates only this rule and finds from *entity & relationship* database that user *U3* does not have a leader role and is non-mutual *NMu* of user *U1* in activity *A1*. As all the conditions in this negative rule are true so the request is denied. If this rule were a regular rule instead of exceptional then the request would be allowed by *priority evaluator* using the fact that team entity rule has higher priority than enterprise entity rule. Conflict handling component is used only when two regular or two exceptional rules exist whose *subject* contains same entity.

C. Evaluation

We evaluate our approach using scenario described in Figure 2. Here users are sharing context with other users based on their roles and relationships. We argue that our hybrid sharing control policy containing both user-defined rules and enterprise-defined RBAC rules is better than only using enterprise-defined RBAC policy in providing better sharing and privacy to user's context information.

It can be better demonstrated by comparison of both policies. Two sets of policies are designed for experiment each

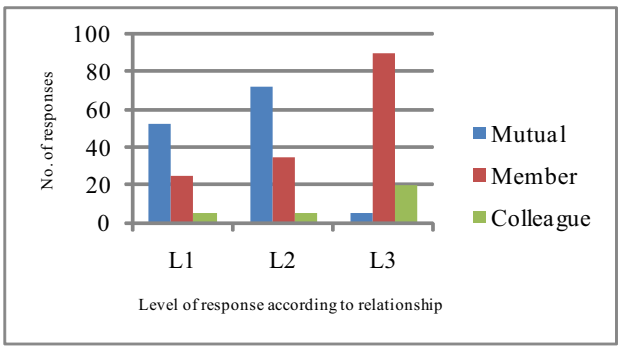

Fig. 4. Sharing requests evaluation with enterprise-defined RBAC policy

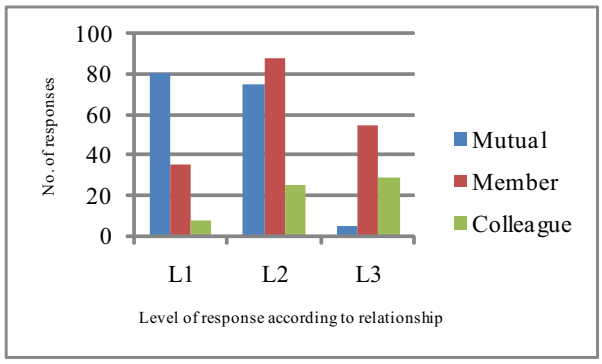

Fig. 5. Sharing requests evaluation with hybrid sharing control policy

based on one of the two policy types. The enterprise-defined RBAC rules are rather static in nature based on used roles only, are created by an administrator. User-defined rules are based on knowledge and relationship of user with other entities like user, activity, team, and enterprise. These rules contain positive, negative as well as exceptional rules which shows user's personal liking or disliking for others. Queries are defined for each user in our scenario where each query is different from other based on role, relationship, or context condition. Queries are evaluated based on two policy sets separately and responses are collected. The level of response for each relationship is calculated where *L1* described the highest level of granularity of the context being shared and *L3* being the lowest. Response from peers using enterprise-defined RBAC policy is shown in Figure 4 while response using our hybrid policy is shown in Figure 5.

It can be seen that results of enterprise-defined policy are rather static in nature with less number of variations (for example, in Figure 4 the values of almost all relationships gradually increase from *L1* to *L3* except *L3* for mutual) while the values for each relationship in hybrid policy graphs vary considerably (for example, in Figure 5 values of *mutual* decrease from *L1* to *L3*, while in other cases they vary considerably). It shows that the user-based policy may vary with time depending on the personal relationship of user with other entities. Also it shows that for closer relationships like "mutual" (which are based on smaller entities), level of detail of response is higher than other relationships (for example *L1* and *L2* in case of Mutual in Figure 5). It is also worth mentioning that total number of response is higher in case of hybrid policy which illustrates the fact that user-defined hybrid policy tends to share more context with others while preserving it privacy to certain level of context with each entity.

VI. CONCLUSION AND FUTURE WORK

The paper describes hybrid sharing control model, its implementation and evaluation using an example from dynamic collaborative environment. Due to complexity of number of involved entities in system, policy conflict can often occur which are analyzed and a mechanism is defined for priorities and conflict handling. Architecture and implementation of the system are described with a running example and evaluation of the system is described by comparing two types of policies using number of responses and level of response for each type of relationship which proves the effectiveness of our hybrid policy. Future work includes the use of semantic technologies to enhance the description of entities and policy rules.

ACKNOWLEDGEMENT

This work is partially supported by the European Union through the FP7-216256 project COIN.

REFERENCES

[1] K. Smith, L. Seligman, and V. Swarup, "Everybody share: The challenge of data-sharing systems," *IEEE Computer*, vol. 41, no. 9, pp. 54–61, 2008.
[2] "European union project coin," http://www.coin-ip.eu.
[3] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.
[4] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino, "A unified framework for enforcing multiple access control policies," *SIGMOD Rec.*, vol. 26, no. 2, pp. 474–485, 1997.
[5] W. Tolone, G. J. Ahn, and T. pai, "Access control in collaborative systems," ACM Survey, 2005.
[6] G. Ahn, "Authorization management for role based collaboration," IEEE int. conf. on System, Man and Cybernetic, Washington, 2003.
[7] M. Convington, "Securing context aware applications using environment roles," ACM, VA, 2001.
[8] S.-H. Park, Y.-J. Han, and T.-M. Chung, "Context-role based access control for context-aware application," HPCC 2006, pp. 572-580, Springer-Verlag Berlin Heidelberg 2006.
[9] V. Kapsalis, L. Hadellis, D. Karelis, and S. Koubias, "A dynamic context-aware access control architecture for e-services," computers and security 25(2006)507-521.
[10] A. K. Malik, H.-L. Truong, and S. Dustdar, "Dyscon: Dynamic sharing control for distributed team collaboration in networked enterprises," in *CEC '09*, 2009, pp. 279–284.
[11] C. Groba, S. Gross, and T. Springer, "Context dependent access control for contextual information," IEEE ARES, 2007.
[12] A. K. Malik and S. Dustdar, "Context-aware sharing control using hybrid roles in inter-enterprise collaboration," ICSOFT 2010, Athens, Greece, 22-24 July, 2010.
[13] E. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," *IEEE Transactions on Software Engineering*, vol. 25, no. 6, pp. 852–869, 1999.
[14] F. Cuppens, N. Cuppens-Boulahia, and M. B. Ghorbel, "High level conflict management strategies in advanced access control models," *Electron. Notes Theor. Comput. Sci.*, vol. 186, pp. 3–26, 2007.
[15] N. Dunlop, J. Indulska, and K. Raymond, "Dynamic conflict detection in policy-based management systems," in *EDOC '02.*, 2002, pp. 15–26.