Secure and Conditional Resource Coordination for Successful Collaborations

Dongxi Liu, Surya Nepal, David Moreland, Shiping Chen, Chen Wang, and John Zic

Networking Technologies Lab, CSIRO ICT Center, Marsfield, NSW 2122, Australia
firstname.lastname@csiro.au

Abstract. Successful completion of collaborations is necessary for collaborating participants to achieve their prescribed collaboration purposes. In this paper, we address the problem of successful completion of collaborations under a new model, called collaborative resource model. This model is graph-based, allowing participants to describe different ways to contribute and require resources for collaborations and the dependency relations among these resources. Resources in this model are protected by access control policies declared not only by resource providers but also by resource requestors. The requestors policies state how they will redistribute the acquired resources and thus increase the confidence of the providers to share resources. Except access control policies, resources are also constrained by usage conditions to reflect the fact, for instance, that a resource might be available only at some time. Based on this model, we present a coordination mechanism. Successful coordination means that all participants can get the necessary resources to complete their collaborations.

Keywords: Secure Resource Model, Robust Collaboration, Resource Coordination, Access Policies.

1 Introduction

Recently, a communication trend has clearly emerged for resources to be able to dynamically collaborate with each other within a distributed environment for various application scenarios, such as health, education and emergencies. This type of *collaborative environment* is rapidly becoming a commonplace means for enabling a collective of entities to deliver significantly bigger, better and more beneficial outcomes than they otherwise could by themselves. All collaborations are established for certain beneficial purposes. In order to achieve these purposes, it is necessary to guarantee the collaborations can be completed successfully by all participants. In particular for mission-critical collaborations, such as collaborative medical operations and collaborative military actions, the failure of such collaborations may cause great damages or losses.

The failures of collaboration can be caused by various reasons based on the fact that collaborations always involve resource sharing among participants with particular business protocols. For example, a failure can happen if one participant uses a protocol mismatched with the protocol used by other participants [1,2]. However, in this paper, we focus on the reason where some participants fail to get the necessary resources for continuing the current collaboration either because the resources are not available or

E. Bertino and J.B.D. Joshi (Eds.): CollaborateCom 2008, LNICST 10, pp. 287–303, 2009.

they do not have the privileges to access the resources. For example, suppose an operation is being collaborated by doctors from several hospitals. If during this operation one doctor from outside needs to access the medical records managed by the local hospital, but without enough rights, then the operation may have to get stopped. This will be a disaster for the patient.

In this paper, we first propose a collaborative resource model, and then based on this model we give a secure resource coordination mechanism. Before a collaboration begins, the involved resources are coordinated. A successful coordination guarantees that all participants can obtain the necessary resources to complete their collaboration.

1.1 Overview of the Collaborative Resource Model

In our collaborative resource model, participants not only contribute resources, but also require resources from other participants to complete collaborations. The resources contributed or required may be dependent on each other and the model allows participants to express such dependency relations among resources. For example, participants can express in this model what resources they will contribute according to what resources they can acquire from other participants. There are different reasons for this: the contributed resources cannot function correctly unless the dependent resources can be acquired, or participants do not feel fair if they think they contribute valuable resources, but others do not. For another example, when some resource is offered, a participant may want to offer another resource because these two resources must be used together and the second resource is dependent on the first one. This model is graph-based so that the dependency relations among resources can be captured naturally. Collaborations based on this model are said resource centric.

On the other hand, even though participants would like to contribute resources for collaborations, it does not mean that they would like every participant to access their resources freely since some participants may be competitors. Our model allows participants to protect their resources by specifying certificate-based policies and the authorization is based on the certificates the resource requestor can provide. This access control mechanism follows the widely accepted principle of trust management [3,4] for the distributed environment where participants belong to different administrative domains and maybe do not know each other in advance. It is worth mentioning that in our access control mechanism not only the resource providers but also the resource requestors can or need to define access control policies. For example, when a hospital requires the medical records from a patient, it needs to specify policies to show who in the hospital can access these data, and the patient releases the medical records only when the policies of the hospital are strong enough for the patient.

Another aspect about the collaborative resource model is that each resource has some conditions to restrict the usage of resources. For example, in a dynamic collaboration, a resource is available only from Monday to Tuesday, and the usage condition of this resource can help reflect this fact.

1.2 Overview of the Resource Coordination Mechanism

For a collaboration, we hope all participants can get their needed resource, so that the collaboration is robust and can complete successfully. A collaboration becomes

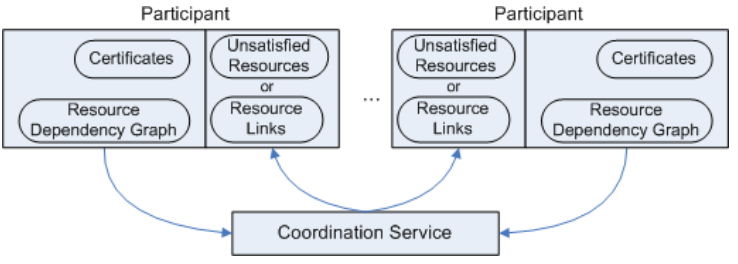

Fig. 1. The Framework of Resource Coordination

complex if it involves a lot of participants and the resources contributed and required by each participant have complex dependency relations and are protected and restricted by complex policies and usage conditions. For a complex collaboration, it is not straightforward to check manually whether this collaboration can complete successfully or not. To address this problem, we give an automatic resource coordination mechanism based on the collaborative resource model, shown in Figure 1. The coordination service is used to check whether the resource requirements of all participants are satisfied before collaborations begin. The coordination service takes as input the certificates and resource descriptions of all participants. As introduced before, the resources of a participant is described in a graph. If the coordination succeeds, a set of resource links is returned to each participant. A resource link connects a resource requirement to a set of available resources, implying that the requestor has enough rights to access the resources and the usage conditions of these resources are satisfied by the requestor. For example, a resource link may state that a requirement for a special medical device can be satisfied by two such devices, each covering different time periods. Resource links can be incorporated into eContracts [5,6] to express resource agreements among participants. If the coordination fails, the coordination service specifies the unsatisfied resources to the participants requiring such resources. However, this paper focuses on the coordination mechanism and algorithm, and how to effectively use resource links as agreements and how to resolve the coordination failure are our future work.

The remainder of this paper is organized as follows. In Section 2, we introduce a scenario to motivate the resource centric collaborations. The collaborative resource model is introduced in Section 3 and a coordination mechanism based on state-space exploration is presented in Section 4. Section 5 gives an implementation algorithm for the coordination mechanism. Section 6 discusses the related work and finally Section 7 concludes the paper.

2 A Scenario for Resource Centric Collaboration

The collaborations we concern here are resource centric in the sense that all collaborating participants focus on what resources they will provide and what resources they can get from other participants to achieve their collaboration purposes. The following scenario illustrates this kind of collaborations.

Suppose in a collaborative medial diagnosis, the participants include a patient, a hospital and a medical lab. The patient has a lot of medical records, some of which are sensitive; the hospital would like to appoint senior or junior doctors for this collaboration; the medical lab can provide expensive equipments to process medical data, and different equipments have different features and may be available at different time. That is, every participant has multiple choices to contribute resources for the collaboration. For a participant, what resources to contribute may depend on the resources available from other participants. For example, the patient would like to release his sensitive medical records if he is treated by a senior doctor, otherwise he wants to provide only the insensitive medical records; the hospital would like to assign a senior doctor if the medical lab can offer an equipment with advanced features at the working time of the senior doctor, otherwise a junior doctor is assigned; the medical lab provides its equipments according to what data to be processed. The resources shared in collaboration are protected by security policies and restricted by usage constraints. For example, the patient wants to protect his sensitive medical records before and even after these data are released to the hospital. The usage constrains of resources impose conditions of using the resources. For example, the medical lab may ask an equipment must be run under certain temperature.

3 The Collaborative Resource Model

In this section, we present the collaborative resource model. A collaboration is always carried on by participants. In our model, these participants collaboratively contribute or require resources to achieve their collaboration purposes. The resources contributed or required by a participant may have dependency relations.

The model is compromised of a set of tuples, each of which describes a participant. Each tuple has the form of $(p, Cert, ResG)$, where p is a participant, *Cert* a set of certificate instances describing the profile of p, and $ResG$ a graph describing the resources contributed or required by p and the dependency relations among these resources. Note that a participant p in our work can be an individual or an organization, and if p is an organization, it can redistribute the acquired resources to its members. In what follows, we will describe the certificates and the resource dependency graph in this model.

3.1 Certificates and Certificate Patterns

Collaborations may occur among participants belonging to different administrative domains. In order to regulate access to resources in collaborations, we should make authorization decisions according to requestors' attributes [3,4]. As usual, this model uses certificates to characterize the attributes of participants.

A certificate is defined with the form $cert^p(p')$, meaning that the principal p (may or may not be a participant in the current collaboration) issues the certificate $cert$ about the participant p'. For example, the certificate $\texttt{doctor}^{\mathrm{H}}(\mathrm{Tom})$ means that Tom is a doctor certified by the hospital H; the certificate $\texttt{nurse}^{\mathrm{Tom}}(\mathrm{Alice})$ means that Tom certifies Alice as his nurse. Certificates can be extended to include more information, such as Alice's address and telephone. For brevity, we ignore such extensions in this work.

Certificates represent certified attributes of principals, as shown by the above examples. They are however too rigid when used to express access policies (defined later). For example, if we want to formulate a policy to allow any doctor to access some confidential data, then we have to explicitly grant each doctor this privilege based on his or her doctor certificate. In this model, we use certificate patterns (like parameterized roles [4]) to make policy definition more convenient.

For a certificate $cert^p(p')$, if any of its parts *cert*, p or p' is replaced by a variable (represented by x, y or z), then it becomes a certificate pattern. A certificate pattern denotes a set of certificates which match this pattern. For example, the certificate $\texttt{doctor}^{\mathrm{H}}(\mathrm{Tom})$ matches the pattern $\texttt{doctor}^{\mathrm{H}}(x)$, which means any doctor x certified by the hospital H, and also matches the pattern $\texttt{doctor}^{y}(x)$, which means any doctor x certified by any hospital y.

Certificate patterns can be ordered in terms of the number of certificates they can include. The order relation of certificate patterns will be used later to check whether the policies specified by the resource requestors are stronger than the policies given by the resource providers. The largest certificate pattern is $z^y(x)$, which can be matched by any certificate, and a certificate is also a pattern containing only itself. In the following definition, three placeholders $\square$, $\odot$ and $\boxdot$ are used to describe the shape of certificate patterns, where $\square$ is either a certificate or a variable, and $\odot$ and $\boxdot$ are either a participant or a variable. For simplicity, we assume $\square$, $\odot$ and $\boxdot$, if they are variables, are different variables in a certificate pattern.

Definition 1 (Order of Certificate Patterns). *Suppose CP_1 and CP_2 are two certificate patterns with the shapes $\square^{\odot}(\boxdot)$ and $\square'^{\odot'}(\boxdot')$, respectively. We say CP_2 is larger than CP_1, written as $CP_1 \sqsubseteq_\Theta CP_2$, if $\square \sqsubseteq_{\Theta_1} \square'$, $\odot \sqsubseteq_{\Theta_2} \odot'$ and $\boxdot \sqsubseteq_{\Theta_3} \boxdot'$, with $\Theta = \Theta_1 \cup \Theta_2 \cup \Theta_3$. The relation $\sqsubseteq_\Theta$ are determined by the following rules.*

- *cert* $\sqsubseteq_\emptyset$ *cert, or cert* $\sqsubseteq_{\{x/cert\}}$ x*;*
- $p \sqsubseteq_\emptyset p$*, or* $p \sqsubseteq_{\{x/p\}} x$*;*
- $y \sqsubseteq_{\{x/y\}} x$.

In the above definition, Θ is a set of substitutions generated when comparing two patterns. The substitution $x/cert$ means the variable x is to be replaced with *cert*, and similarly for the substitutions x/p and x/y. The substitutions will be used in Section 3.3 when comparing two policies. A policy is defined with a set of certificate patterns, and using substitutions can help check whether the link information captured by the same variables among patterns in the policy of resource providers is also kept by certificate patterns in the policy of resource requestors. For example, a policy consisting of patterns $\{\texttt{nurse}^x(y), \texttt{doctor}^{\mathrm{H}}(x)\}$ means that y is the nurse of x who is a doctor of the hospital H. When it is compared with a policy containing patterns $\{\texttt{nurse}^{\mathrm{Tom}}(y), \texttt{doctor}^{H}(\mathrm{Tom})\}$, we obtain $\texttt{nurse}^{\mathrm{Tom}}(y) \sqsubseteq_{\{x/\mathrm{Tom},y/y\}} \texttt{nurse}^x(y)$ and $\texttt{doctor}^{\mathrm{H}}(\mathrm{Tom}) \sqsubseteq_{\{x/\mathrm{Tom}\}} \texttt{dcotor}^{\mathrm{H}}(x)$, which indicate that the link information captured by x is kept by the second policy since both x are replaced with the same value Tom in the second policy. In later sections, we will not distinguish between certificates and certificate patterns; when we want to mean certificates we say certificate instances.

3.2 Resource Dependency Graph

Participants always need to contribute resources and at the same time require resources from other participants to achieve their collaboration purposes. The resources a participant would like to contribute or require may depend on what resources he can contribute or acquire during collaborations. For example, in a collaboration between a patient and a hospital about medical consultation, the patient may contribute all his medical records if a medical expert is appointed for his consultation, or he may contribute only a part of his medical records if a doctor just graduated is appointed.

The usage of resources, whether they are being contributed or required, is controlled by access polices and conditions in our model. An interesting point is that when participants require resources they also need to specify policies for these resources. These policies tell the resource provider how these resources will be used by the requestor (and its members if the requestor is an organization). For example, a hospital may specify a policy that says only senior doctors are permitted to access the medical records acquired from a patient. In order to get a resource from a provider, the requestor's policies must be stronger than the the provider's policies. This increases the confidence of participants about how their resources are used by other participant during collaborations.

In our model, the resources contributed and required by a participant, their dependency relations, and their access policies and conditions are all captured by the resource dependency graph, which is a directed acyclic graph. Before defining resource dependency graphs, we first describe the resources in our model.

A resource is specified by a pair (r, Att), where r is the name of the resource and Att is a set of attributes characterizing the resource. An attribute has a name and a value. For example, a resource `doctor` could have a `position` attribute with the value `senior`. Attributes are needed when checking whether an offered resource matches the required resource. We use R to denote the pair (r, Att). The operator $\sqsubseteq$ is overloaded to reresent resource match (and other order relations later).

Definition 2 (Resource Match). *Let* $R_1 = (r_1, Att_1)$ *and* $R_2 = (r_2, Att_2)$. *If* $r_2 = r_1$ *and* $Att_2 \subseteq Att_1$, *then* R_1 *matches* R_2, *denoted as* $R_2 \sqsubseteq R_1$.

A resource dependency graph $ResG$ is described by a pair $(N, \rightarrow)$, where N is a set of nodes, and $\rightarrow$ is a set of directed edges. The details of nodes and edges are given below.

A node $n \in N$ has two forms $(-, R, Pol, con)$, or $(+, R, Pol, con)$. That is, n is either a negative node or a positive node. The negative node means that the resource R is being required, while the positive node means R is being contributed. The policies Pol are used to protect the required or contributed resources, and the condition *con* specifies the condition of using the resource. The polices and conditions are introduced in the next two sections, respectively. For a node n, we write $\texttt{sign}(n) = -$ or $\texttt{sign}(n) = +$, depending on whether n is a negative node or a positive node. For each resource dependency graph, we specify a special node `sta` as the start node and a special node `end` as the end node. The unique start and end nodes make it convenient later to define the initial and the final states for the coordinating mechanism.

An edge $(n_1, n_2) \in \rightarrow$ means that only after the resource described in n_1 is contributed or acquired, the participant would like to contribute or needs to require the resource in n_2. We also write $n_1 \rightarrow n_2$ for $(n_1, n_2) \in \rightarrow$. For example, if $\texttt{sign}(n_1) = -$

and $\texttt{sign}(n_2) = +$, the edge $n_1 \rightarrow n_2$ means the participant first needs a resource in n_1, and then contributes the resource in n_2. Note that positive nodes and negative nodes in a resource dependency graph do not necessarily appear alternately. For example, if $\texttt{sign}(n_1) = -$ and $\texttt{sign}(n_2) = -$, the edge $n_1 \rightarrow n_2$ means the participant first wants a resource in n_1, and then another resource in n_2.

A node n may have more than one child nodes and parent nodes. We write $\texttt{cnodes}(n)$ for the set of child nodes of n, and $\texttt{pnodes}(n)$ for the set of its parent nodes. Suppose $\texttt{cnodes}(n) = \{n_1, ..., n_m\}$. Then after the resource in n is contributed or required, any node n_i $(1 \leq i \leq m)$ can be chosen to let the collaboration proceed. That is, the edges $n \rightarrow n_i$ $(1 \leq i \leq m)$ are mutually exclusive for one collaboration; choosing different edges implies collaborations with different resources involved. Similarly, if $\texttt{pnodes}(n) = \{n_1, ..., n_m\}$, then the resource in n is processed when any parent node $n_i (1 \leq i \leq m)$ is chosen to provide or require resources. Therefore, each path in a resource dependency graph represents a possible way of contributing and requiring resources for a participant in a collaboration.

3.3 Access Polices

In our model, resources, whether they are being contributed or being required, are all protected by a set of policies *Pol*, specified in each node of resource graphs. In the following, we assume the policies *Pol* is specified in a node n for the resource R.

A policy $pol \in Pol$ has the form $Cert \rightarrow (op, Cert')$, meaning that the participants with certificates $Cert$ have the right to access the resource R and after getting R they can delegate the operation op on R to users having certificates $Cert'$. If $Cert$ and $Cert'$ are the same certificates, then the participants getting and using the resources are the same. This special case corresponds to the traditional access policies, where the authorized users get and use resource by themselves. One operation *op* may be protected by more than one policies in *Pol* for different authorization cases. A resource user can execute *op* on R if there exists a policy $pol \in Pol$ that authorizes the user for this operation. For example, suppose a patient defines the following two policies in the policy set *Pol* to protect his medical records:

$$\{\texttt{hospital}^{\texttt{Gov}}(\texttt{H})\} \rightarrow (\texttt{read}, \{\texttt{doctor}^{\texttt{H}}(x)\})$$
$$\{\texttt{hospital}^{\texttt{Gov}}(\texttt{H})\} \rightarrow (\texttt{read}, \{\texttt{nurse}^{x}(y), \texttt{doctor}^{\texttt{H}}(x), \texttt{senior}^{\texttt{H}}(x)\})$$

The first policy means the hospital H can access the medical records and let its doctors to read these data, and similarly for the second policy.

When a participant obtains resources during collaboration, this participant may redistribute this resource to other users. This case is common when the participant is an organization, such as a hospital, and it needs its affiliated members to use the resources to carry on the current collaboration, as shown in Figure 2. Hence, in order to let resource providers be willing to release their resources for successful collaborations, the resource requestors must have not only enough certificates to get themselves authorized, but also policies stronger than the polices desired by resource providers. Thus, if a user can provide certificates to access resources from the participant who is redistributing resources, then the certificates provided by this user are also enough to pass the policies of the resource provider. A stronger policy is defined with a set of stronger certificates, which is defined below.

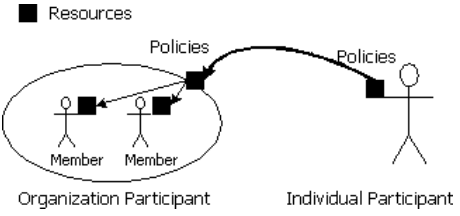

Fig. 2. Redistribution of Acquired Resources

Definition 3 (Order of Certificate Sets). *Suppose* $Cert_1$ *and* $Cert_2$ *are two sets of certificates. We say* $Cert_1$ *is stronger than* $Cert_2$*, written as* $Cert_1 \sqsubseteq Cert_2$*, if the following two conditions hold.*

- $Cert_2 = \emptyset$*;*
- *Let* $Cert_2 = \{cert_2\} \cup Cert_2'$*. Then, there exists* $cert_1 \in Cert_1$*, such that* $cert_1 \sqsubseteq_\Theta cert_2$ *and* $Cert_1' \sqsubseteq \Theta(Cert_2')$*, where* $Cert_1' = Cert_1 \setminus \{cert_1\}$*.*

In the above definition, the notation $\Theta(Cert_2')$ denotes a set of certificates obtained by applying all substitutions in Θ to all certificates in $Cert_2'$.

Definition 4 (Order of Policies). *Suppose* pol_1 *and* pol_2 *are two policies, and Cert is a set of certificate instances. Let* $pol_1 = Cert_1 \rightarrow (op_1, Cert_1')$ *and* $pol_2 = Cert_2 \rightarrow (op_2, Cert_2')$*. If* $op_1 = op_2$*,* $Cert_1 \sqsubseteq Cert_2$*,* $Cert \sqsubseteq Cert_2$ *and* $Cert_1' \sqsubseteq Cert_2'$*, then* pol_1 *is stronger than* pol_2 *when enhanced with Cert, written as* $pol_1 \sqsubseteq_{Cert} pol_2$*.*

In the relation $pol_1 \sqsubseteq_{Cert} pol_2$, pol_1 and pol_2 are policies specified by the resource requestor and providers, respectively, and $Cert$ is the profile certificates of the requestor. Two policies are comparable only when they both concern the same operation.

In order to get the needed resource, each policy specified by resource requestors must have a corresponding policy specified by resource providers, with the former stronger than the latter, so that resource providers have the confidence that their resource will be redistributed in ways not violating their polices. Two sets of policies are compared by the following definition.

Definition 5 (Order of Policy Sets). *Suppose* Pol_1 *and* Pol_2 *are two sets of policies, and Cert is a set of certificate instances. If for every* $pol_1 \in Pol_1$ *there exists* $pol_2 \in Pol_2$*, such that* $pol_1 \sqsubseteq_{Cert} pol_2$*, then* Pol_1 *is stronger than* Pol_2 *when enhanced with Cert, written as* $Pol_1 \sqsubseteq_{Cert} Pol_2$*.*

Continuing with the above example, we suppose the patient wants to collaborate with the hospital H. The hospital knows Tom is a doctor and owns the certificate $\texttt{hospital}^{\texttt{Gov}}(\mathrm{H})$, and defines the following policies to protect the medical records of the patient:

$$\texttt{hospital}^{\texttt{Gov}}(\mathrm{H}) \rightarrow (\texttt{read}, \{\texttt{doctor}^{\mathrm{H}}(\mathrm{Tom})\})$$
$$\texttt{hospital}^{\texttt{Gov}}(\mathrm{H}) \rightarrow (\texttt{read}, \{\texttt{nurse}^{\mathrm{Tom}}(y), \texttt{doctor}^{\mathrm{H}}(\mathrm{Tom}), \texttt{senior}^{\mathrm{H}}(\mathrm{Tom})\})$$

According to the above definition, these policies are stronger than the policies specified by the patient since they only allow the doctor Tom and the nurse of Tom to access the patient's medical records. And when the hospital acquires the medical records, it does not release to other organizations as specified in these polcies.

3.4 Access Conditions

In our model, each resource is attached with a condition *con*, which imposes restrictions to resource accesses based on collaboration context parameters. For example, a condition may state that a resource is accessible between Monday and Tuesday or on Friday. A condition is a logical formula. `True` is a condition that is always satisfied, and `False` is a condition that cannot be satisfied. So if a resource is attached with `True`, then this resource can be used under any condition; if it is attached with `False`, then it actually cannot be used by anybody.

We use a differentiation operator `diff` to manipulate conditions. Given two conditions, con_1 and con_2, the operation $\texttt{diff}(con_1, con_2)$ returns a new condition con, meaning that if the condition con_2 holds, then con implies con_1. The definition of `diff` depends on particular collaboration contexts. However, the following laws hold in all collaboration contexts:

$$\begin{aligned}
&\texttt{diff}(con, con) && = \texttt{True}, \text{if } con \neq \texttt{False}\\
&\texttt{diff}(con \vee con', con'') && = \texttt{diff}(con, con'') \vee \texttt{diff}(con', con'')\\
&\texttt{diff}(con \wedge con', con'') && = \texttt{diff}(con, con'') \wedge \texttt{diff}(con', con'')\\
&\texttt{diff}(\texttt{False}, con) && = \texttt{False}\\
&\texttt{diff}(con, \texttt{False}) && = con\\
&\texttt{diff}(con, \texttt{True}) && = \texttt{True}, \text{if } con \neq \texttt{False}
\end{aligned}$$

For example, in a collaboration, suppose a participant needs a device from 10am to 5pm (con_1). If such a device is offered with the condition `True` (i.e., it can be used anytime), then $\texttt{diff}(con_1, \texttt{True})$ returns `True`, meaning that the participant's requirement can be satisfied by this offer; if such a device can be used from 11am to 1pm (con_2), then $\texttt{diff}(con_1, con_2)$ returns a conjunctive condition, specifying the time periods from 10am to 11am and from 1pm to 5pm. That is, if there are the same devices available during these two periods, the participant's requirement can be satisfied. Note that not all conditions can be divided. For example, if a condition specifies a temperature of 1000°C, we may not divide it into two 500°Cs. If a condition con_1 is indivisible, then $\texttt{diff}(con_1, con_2)$ simply returns con_1 if con_2 cannot cover con_1. That is, the availability of resources with the condition con_2 has nothing to do with the requirement of the same resources with the condition con_1.

3.5 Resource Satisfaction

For a successful collaboration, all participants must be able to get their needed resources. In the following, we introduce step by step what resource satisfaction means for collaborations in our model.

A collaboration generally involves many resource requests from all participants. Here, we starts from the satisfaction problem of a single request. If a request of one participant is possible to be satisfied, then there must be other participants who contribute the requested resources. The predicate `support` defined below captures the necessary conditions on a set of positive nodes for satisfying a request in a negative node.

Definition 6 (Support Nodes). *Given a negative node n and a set of certificate instances Cert, a set of nodes $N = \{n_1, ..., n_k\}$ is a set of support nodes of n with respect to Cert, written as* $\texttt{support}(N, n, Cert)$*, if the following conditions hold:*

- $\texttt{sign}(n') = +$ *for all* $n' \in N$*;*
- $n.R \sqsubseteq n'.R$ *for all* $n' \in N$*;*
- $n.Pol \sqsubseteq_{Cert} n'.Pol$ *for all* $n' \in N$*;*
- $\texttt{diff}(n.con, n_1.con \vee ... \vee n_k.con) = \texttt{True}$.

Having support nodes does not mean a negative node can be satisfied definitely. This is because its support nodes may have negative parent or ancestor nodes that need to be satisfied first. Recall that a path in a resource graph represents a possible way to contribute and require resources for a participant in a collaboration. A well supported node n defined coinductively below asks every support node n' to be in a path from a resource graph with all negative parent and ancestor nodes also well supported. Thus, a well supported node can be satisfied definitely by its support nodes. In the definition below, $\texttt{pnodes}^+$ means the transitive closure of pnodes. Note that a node has only one parent node on a path.

Definition 7 (Well Supported Nodes). *Given a set of paths P from resource graphs and a set of certificate instances $Cert$, a negative node n is well supported in the set P with respect to $Cert$ if there exists a set of nodes N from P such that* $\texttt{support}(N, n, Cert)$ *and the following conditions hold for all* $n' \in N$.

- *for all* $n'' \in \texttt{pnodes}^+(n')$*, if* $\texttt{sign}(n'') = -$*, then n'' is well supported.*

Finally, based on the concept of well supported nodes, the resource satisfaction for collaborations in our model is defined below.

Definition 8 (Resource Satisfaction). *Suppose a collaboration consists of m participants $(p_i, Cert_i, ResG_i)$ $(1 \leq i \leq m)$. The resource requests in this collaboration are satisfiable if there exists a set of paths $\{path_i | 1 \leq i \leq m\}$, where $path_i$ comes from the graph $ResG_i$ with* sta *and* end *as its start and end nodes respectively, such that for all j $(1 \leq j \leq m)$ all negative nodes on $path_j$ are well supported in the set $\{path_i | 1 \leq i \leq m\}$ with respect to $Cert_j$.*

The following two sections will give the mechanism and algorithm to find the paths in a set of resource graphs that all have well supported negative nodes.

4 Coordination Based on State-Space Exploration

In this section, we present the coordinating mechanism to check whether collaborations in our model can be completed successfully or not. In our model, the reason for collaboration failures is that some participants fail to obtain the resources necessary for continuing the current collaboration. Since a resource in our model may be contributed depending on the availability of other resources and protected by access policies and conditions, we cannot simply say it is available to other participants because of its positive presence in some resource graphs. Our coordination mechanism is based on state-space exploration, which is widely used for computer-aided verification. A state represents a resource coordinating status, like what resources have been contributed and what resource requirements have been satisfied or are being proposed. A state transition indicates the coordination enters a new state because a resource is contributed or a resource requirement is satisfied.

4.1 States

A coordinating state has the form (RB, PS, RL), where RB is the resource base containing all resources currently offered by participants, PS describes the resource status of each participant and RL means resource links indicating how resource requirements are satisfied at this state. An element in *RB* has the form (p, n), where $\texttt{sign}(n) = +$, meaning that the participant p contributes the resource described by the node n in $p's$ resource dependency graph. The status of each participant in PS is also described by the pair (p, n), where $\texttt{sign}(n) = +$ or $\texttt{sign}(n) = -$, meaning that p is waiting to contribute or require the resource in n, a node in $p's$ resource dependency graph. Note that we can move to the children nodes of n by using the `cnodes` operator when changing $p's$ status. A resource link in RL has the form $((p, n), \{(p_1, n_1), ..., (p_m, n_m)\})$, where $\texttt{sign}(n) = -$ and $\texttt{sign}(n_i) = + \ (1 \leq i \leq m)$, meaning that the resource requirement n of p can be satisfied by combining the resources n_1,..., n_m offered by participants p_1,..., p_m. For example, suppose p wants a device from 1pm to 5 pm. Then this requirement is satisfiable if p_1 and p_2 can collaboratively provide the device from 1pm to 2pm and from 2pm to 5pm, respectively.

The coordination procedure starts from an initial state. Suppose a model consists of $(p_i, Cert_i, ResG_i) \ (1 \leq i \leq m)$ for m participants. Then the initial state for this model is $(\emptyset, \{(p_1, \texttt{sta}), ..., (p_m, \texttt{sta})\}, \emptyset)$. Recall that every resource graph $ResG_i$ starts with the special node `sta`. In the initial state, each participant has not begun to contribute and require resources, so there is no contributed resources, nor resource links.

For a collaboration, if the resources involved cannot be coordinated to a final state, then this collaboration will not be possible to complete. A final state has the form $(RB, \{(p_1, \texttt{end}), ..., (p_m, \texttt{end})\}, RL)$, where no participant has resources to contribute and require. The resource links RL contain the information of how to satisfy all resource requirements in collaborations. This information can be used at collaboration time to route resource requests. Hence, in a final state, if a resource appears in RB but not referred to in RL, then this resource is redundant for this collaboration and can be removed safely.

4.2 State Transitions

The state transitions of coordination are caused by the status change of participants. For example, a state changes when a participant contributes a resource or have a resource request satisfied. To help define how state transitions occur, the operator `reduce` (RB, p, n) in Figure 3 is used to check whether the resource request in the negative node n from the resource graph of p can be satisfied by the resource base RB. In this definition, $\texttt{certs}(p)$ means the certificate instances for the profile of p. This operator returns a pair with its first component being a node or a special value ϵ and its second component being a set of nodes. If the first component is ϵ, then it means that the requested resource of n can be satisfied and the second component contains the support nodes of n, otherwise the first component is a node whose satisfaction implies the satisfaction of n.

$$\texttt{reduce}(\emptyset, p, n) = (n, \emptyset)$$

$$\texttt{reduce}(\{(p', n')\} \cup \texttt{RB}, p, n) = \begin{cases} \texttt{reduce}(\texttt{RB}, p, n) & \text{if } n.R \not\sqsubseteq n'.R \text{ or } n.Pol \not\sqsubseteq_{\texttt{certs}(p)} n'.Pol \\ \texttt{reduce}(\texttt{RB}, p, n) & \text{if } n.R \sqsubseteq n'.R,\ n.Pol \sqsubseteq_{\texttt{certs}(p)} n'.Pol, \\ & \quad \texttt{diff}(n.con, n'.con) = n.con \\ (n'', \{(p', n')\} \cup S) & \text{if } n.R \sqsubseteq n'.R,\ n.Pol \sqsubseteq_{\texttt{certs}(p)} n'.Pol, \\ & \quad \texttt{diff}(n.con, n'.con) = con' \text{ and } con' \neq n.con \\ & \quad \text{with } (n'', S) = \texttt{reduce}(\texttt{RB}, p, (-, n.R, n.Pol, con')) \\ (\epsilon, \{(p', n')\}) & \text{if } n.R \sqsubseteq n'.R,\ n.Pol \sqsubseteq_{\texttt{certs}(p)} n'.Pol \\ & \quad \text{and } \texttt{diff}(n.con, n'.con) = \texttt{True} \end{cases}$$

Fig. 3. The `reduce` Operator

Proposition 1. *Given a resource base RB and a node n from the resource graph of participant p with* $\texttt{sign}(n) = -$*, if* $\texttt{reduce}(RB, p, n) = (\epsilon, S)$ *and let* $N = \{n' | (p', n') \in S\}$*, then N is a set of support nodes of n, that is,* $\texttt{support}(N, n, \texttt{certs}(p))$ *holds.*

In the following, we describe first from the perspective of a participant how state transitions occur, and then from the perspective of all participants.

Suppose the current state is $st = (RB, PS, RL)$ and $(p, n) \in PS$. The next states caused by the status change of p, denoted as $\texttt{next}^{(p,n)}(st)$, are generated according to the following rules.

- Rule 1: $n = \texttt{sta}$. $\texttt{next}^{(p,n)}(st) = \{(RB, PS' \cup \{(p, n')\}, RL) | n' \in \texttt{cnodes}(n)\}$, where $PS' = PS \setminus \{(p, n)\}$.
- Rule 2:$\texttt{sign}(n) = +$. $\texttt{next}^{(p,n)}(st) = \{(RB', PS' \cup \{(p, n')\}, RL) | n' \in \texttt{cnodes}(n)\}$, where $RB' = RB \cup \{(p, n)\}$ and $PS' = PS \setminus \{(p, n)\}$.
- Rule 3: $\texttt{sign}(n) = -$, $\texttt{reduce}(RB, p, n) = (n', S)$ and $n' \neq \epsilon$. $\texttt{next}^{(p,n)}(st) = \{st\}$.
- Rule 4: $\texttt{sign}(n) = -$ and $\texttt{reduce}(RB, p, n) = (\epsilon, S)$. $\texttt{next}^{(p,n)}(st) = \{(RB, PS' \cup \{(p, n')\}, RL') | n' \in \texttt{cnodes}(n)\}$, where $RL' = RL \cup \{((p, n), S)\}$ and $PS' = PS \setminus \{(p, n)\}$.
- Rule 5: $n = \texttt{end}$. $\texttt{next}^{(p,n)}(st) = \{st\}$.

For a state $st = (RB, PS, RL)$, its next states from the perspective of all participants, written as $\texttt{next}(st)$, are obtained by composing all next states from the perspective of each participant.

$$\texttt{next}(st) = \bigcup_{(p,n) \in PS} \texttt{next}^{(p,n)}(st)$$

4.3 Correctness

In this section, we show that the correctness of the above coordination mechanism for checking the resource satisfaction in collaborations. For this purpose, we need to extend the use of `next` operator in the following two aspects.

First, the `next` operator is extended to a set of states *ST*.

$$\texttt{next}(ST) = \bigcup_{st \in ST} \texttt{next}(st)$$

Second, the next operator can be applied iteratively many times, say m times, to a state *st* (or a set of states), written as $\texttt{next}^m(st)$.

$$\texttt{next}^1(st) = \texttt{next}(st)$$
$$\texttt{next}^{m+1}(st) = \texttt{next}(\texttt{next}^m(st))$$

Theorem 1 (Correctness of Coordination). *Given an initial state st for a collaboration, there exists some integer m such that $\texttt{next}^{m+1}(st) = \texttt{next}^m(st)$, that is, $\texttt{next}^m(st)$ is a fixed point of the operator* next. *If there is a final state in $\texttt{next}^m(st)$, then the resource requests in this collaboration are satisfiable.*

The next section will give an algorithm to compute the fixed point of the next operator, which is then used to check the resource consistency in collaborations.

5 A Coordination Algorithm

In this section, we present an algorithm that implements the previous coordination mechanism. The main code of the algorithm is SPExplore, shown in Figure 4, which computes the fixed point of the next operator by iteratively calling the subroutine ONext in the same figure. The code ONext implements the transition rules of next operator defined in the previous section but with some optimization. By this optimization, when a negative node is checked against a resource base *RB* to test its satisfaction with the reduce operator, the current check arguments and results are memoized if its request is not satisfied completely by the current resource base *RB*. And then, if *RB* is extended with new contributed resources in the later stage of coordination, it is not needed to check this negative node against the whole resource base, and instead the result of last check is reused and only the newly contributed resources are tested. Hence, this optimization can avoid repeating comparisons of resources, access policies and usage conditions during state transitions.

The optimization used in this implementation is based on the following proposition, which lays the foundation of using the resource base incrementally to test the satisfaction of resource requests.

Proposition 2. *Let $RB = RB_1 \cup RB_2$. For a node n from the resource graph of participant p with $\texttt{sign}(n) = -$, if $\texttt{reduce}(RB, p, n) = (n', S)$, $\texttt{reduce}(RB_1, p, n) = (n_1, S_1)$ and $\texttt{reduce}(RB_2, p, n_1) = (n_2, S_2)$, then $n' = n_2$ and $S = S_1 \cup S_2$.*

In the following, we will introduce how this optimization is implemented. The point of this optimization is to memoize the intermediate check results and reuse them for the further checks when the resource base is extended. The memoization is implemented by a mapping, called *RBMemo* or *RBMemo′* in the code, which maps p and n to (*RB*, (n', S)) if the mapping is defined on p and n. It means that the request in n has been tested against the existing resource *RB* but not satisfied completely, and if the request in n' can be satisfied by some newly contributed resources, then the request in n will be completely satisfied and S is a part of the final support nodes for n. An empty mapping is denoted by $\bullet$, which is undefined for every p and n. The notation *RBMemo*$[(p, n) \mapsto (RB, (n', S))]$ means a new mapping obtained by updating the value

```
Function SPExplore( st)
Input:
   st=(∅, PS, ∅): an initial state
Output:
   WL: a set of states
 begin
    WL = {( st, •)}
    OldWL = WL
    NewWL = ∅
  while OldWL ≠ NewWL do
    NewWL = ∅
    ST = ∅
    for each (st′, RBMemo) ∈ WL do
     AST = ONext(st′, RBMemo)
     NewWL = NewWL ∪ AST
     ST = ST ∪ { st′}
    endfor
    OldWL = WL
    WL = NewWL
  endwhile
  return ST
 end
Function ONext( st, RBMemo)
Input:
   st=(RB, PS, RL): a state
   RBMemo: a mapping from (p, n) to ( RB, (n, S))
Output:
   AST: a set of pairs of state and RB memo
 begin
  AST = ∅
  for each (p, n) ∈ PS do
   PS′ = PS \ (p, n)
   if n = sta then
    ST′ = {( RB, PS′ ∪ {(p, n′)}, RL)|n′ ∈ child(n)}
    AST′ = ST′ × { RBMemo}
   else if sign(n) = + then
    RB′ = RB ∪ {(p, n)}
    ST′ = {( RB′, PS′ ∪ {(p, n′)}, RL)|n′ ∈ child(n)}
    AST′ = ST′ × { RBMemo}
   else if sign(n) = − then
    if RBMemo(p, n) = ( RB′, (n′, S′)) then
     (n′′, S) = reduce(RB\RB′, p, n′)
     if n′′ ≠ ϵ then
      RBMemo′= RBMemo[(p, n) ↦ ( RB, (n′′, S ∪ S′))]
      AST′ = {( st, RBMemo′)}
     else
      RBMemo′= RBMemo[(p, n) ↦ undefined]
      RL′ = RL ∪ {((p, n), S ∪ S′)}
      ST′ = {( RB, PS′ ∪ {(p, n′)}, RL′)|n′ ∈ child(n)}
      AST′ = ST′ × { RBMemo′}
    else if RBMemo(p, n) = undefined then
     if reduce(RB, p, n) = (n′, S)&&n′ ≠ ϵ then
      RBMemo′= RBMemo[(p, n) ↦ ( RB, (n′, S))]
      AST′ = {( st, RBMemo′)}
     else if reduce(RB, p, n) = (ϵ, S) then
      RL′ = RL ∪ {((p, n), S)}
      ST′ = {( RB, PS′ ∪ {(p, n′)}, RL′)|n′ ∈ child(n)}
      AST′ = ST′ × { RBMemo}
   else if n = end then
    AST′ = {(st, RBMemo)}
   endif
   AST = AST ∪ AST′
  endfor
  return AST
 end
```

Fig. 4. The Coordination Algorithm

of (p, n) in *RBMemo* to $(RB, (n', S))$, that is, the new mapping returns $(RB, (n', S))$ for (p, n), and returns $RBMemo(p, n)$ otherwise.

The subroutine `ONext` implements all rules of state transitions for the `next` operator. Only in the rule concerning negative nodes, the mapping *RBMemo* is updated and referenced. For example, for a negative node n of participant p, if (p, n) is mapped to $(RB', (n', S'))$ in *RBMemo*, then the request test is done by the operation reduce$(RB \backslash RB', p, n')$, that is, only part of the whole resource base *RB* is checked and the check works on the node n' from last check. Based on this check, if the request of n is still not satisfied, that is reduce$(RB \backslash RB', p, n') = (n'', S)$ and $n'' \neq \epsilon$, then the mapping *RBMemo* on (p, n) is updated by $RBMemo[(p, n) \mapsto (RB, (n'', S \cup S'))]$ to incorporate the result of this check, otherwise the mapping of (p, n) is changed into `undefined` in *RBMemo*.

6 Related Work

The collaboration failures can also be caused by mismatched business protocols between participants. This problem can be addressed by adapting interactions among participants [1,2]. Our work is complementary to these work since we focus on resource satisfaction for collaborating participants, rather than on protocol consistency.

There are various access control mechanisms for collaborations among different organizations [7,8,9,10]. However, these mechanisms do not consider how they affect the successful completion of collaborations. For example, maybe a participant defines too strong security policies, and hence there are actually no possibility for other participants to access its resources intended for sharing. It is also similar for the work of enhancing business processes with authorization constraints [11]. Two process may collaborate well, but after enhanced with authorization constraints, they probably fail to work together.

The work [12,13] studies how to specify and solve authorization constraints on workflow systems, so as to make sure there are possible successful executions of workflow patterns. Their work focuses on tasks, concerning workflow tasks in the same organization, while our work focuses on resources, coordinating resources from different organizations.

The work [14] proposes an architecture to protect an object (or a resource) by enforcing the object owner's policy in the requestor's platform. This architecture is based on the special hardware for trusted computing. It is not clear whether this architecture can be scaled to the distributed collaboration environments where some participants may not have such special platforms. Access control in our resource has no special requirements on platforms by trusting the requestors faithfully enforce the policies they declare for requiring resources.

As surveyed in [15], graphs can be used to represent the allocation and request status of resources at a system state for checking deadlocks. For that purpose, those graphs have only one kind of nodes representing the resources being allocated and requested. The resources being offered for collaborations cannot be represented in those graphs.

For participants, offering resources can be regarded as their obligations for making collaborations successful. A centralized model has been proposed in [16] to analyze

whether a system state can lead to another system state in which a subject cannot fulfill his obligation within the time window or at the deadline of the obligation. Our collaborative resource model is distributed in the sense that each participant has his own obligations to fulfill and his obligations may affect the ability of other participants to perform their obligations.

7 Conclusion

In this paper, we address the problem of whether collaborations can be completed successfully, so as to achieve the prescribed collaboration purposes. The collaborations we concern here is resource centric and we proposed a collaborative resource model for this kind of collaborations. This model can capture the dependency relations among resources offered or required by participants. This feature gives participants flexible strategies to choose resources for collaborations. Resources in this model are protected by security policies based on certificates and restricted by usage conditions. The certificates based access control mechanism is suitable for participants who do not know each other in advance in a distributed environment. Moreover, this model allows the resource requestors to declare policies, stating how the requestors will redistribute the acquired resources, and thus enables the resource owners to gain more confidence to contribute their resources. The usage condition can be used to model, for instance, dynamic collaborations, where participants may join and leave in the middle of collaborations. Based on the collaborative resource model, we proposed the coordination mechanism. Successful coordination means that there must be a possible way to satisfy the resource requirements of all participants during collaborations.

In the future, we will consider other principles of access control mechanisms, such as separation of duty and binding of duty, and other kind of resources, like consumable resources.

References

1. Nezhad, H.R.M., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW 2007: Proceedings of the 16th International Conference on World Wide Web, pp. 993–1002 (2007)
2. Desai, N., Chopra, A.K., Singh, M.P.: Business process adaptations via protocols. In: SCC 2006: Proceedings of the IEEE International Conference on Services Computing, pp. 103–110. IEEE Computer Society, Los Alamitos (2006)
3. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: Proceedings of the 1996 IEEE Symposium on Security and Privacy, pp. 164–173 (1996)
4. Li, N., Mitchell, J.C., Winsborough, W.H.: Design of a role-based trust-management framework. In: SP 2002: Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp. 114–130. IEEE Computer Society, Los Alamitos (2002)
5. Chan, J., Rogers, G., Agahari, D., Moreland, D., Zic, J.: Enterprise collaborative contexts and their provisioning for secure managed extranets. In: WETICE 2006: Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 313–318. IEEE Computer Society, Los Alamitos (2006)

6. Chan, J., Nepal, S., Moreland, D., Hwang, H., Chen, S., Zic, J.: User-controlled collaborations in the context of trust extended environments. In: WETICE 2007: Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 389–394. IEEE Computer Society, Los Alamitos (2007)
7. Gong, L., Qian, X.: The complexity and composability of secure interoperation. In: Proceedings of the 1994 IEEE Symposium on Security and Privacy, p. 190 (1994)
8. Shehab, M., Bertino, E., Ghafoor, A.: Secure collaboration in mediator-free environments. In: CCS 2005: Proceedings of the 12th ACM Conference on Computer and Communications Security, pp. 58–67 (2005)
9. Warner, J., Atluri, V., Mukkamala, R., Vaidya, J.: Using semantics for automatic enforcement of access control policies among dynamic coalitions. In: SACMAT 2007: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, pp. 235–244 (2007)
10. Zhang, X., Nakae, M., Covington, M.J., Sandhu, R.: A usage-based authorization framework for collaborative computing systems. In: Proceedings of the eleventh ACM symposium on Access Control Models and Technologies, pp. 180–189 (2006)
11. Bertino, E., Crampton, J., Paci, F.: Access control and authorization constraints for ws-bpel. In: ICWS 2006: Proceedings of the IEEE International Conference on Web Services, pp. 275–284 (2006)
12. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM Trans. Inf. Syst. Secur. 2(1), 65–104 (1999)
13. Tan, K., Crampton, J., Gunter, C.A.: The consistency of task-based authorization constraints in workflow systems. In: CSFW 2004: Proceedings of the 17th IEEE workshop on Computer Security Foundations, p. 155. IEEE Computer Society, Los Alamitos (2004)
14. Sandhu, R., Zhang, X.: Peer-to-peer access control architecture using trusted computing technology. In: SACMAT 2005: Proceedings of the tenth ACM symposium on Access control models and technologies, pp. 147–158 (2005)
15. Coffman, E.G., Elphick, M., Shoshani, A.: System deadlocks. ACM Comput. Surv. 3(2), 67–78 (1971)
16. Irwin, K., Yu, T., Winsborough, W.H.: On the modeling and analysis of obligations. In: Proceedings of the 13th ACM Conference on Computer and Communications Security (2006)