From Business Process Choreography to Authorization Policies

Philip Robinson[1], Florian Kerschbaum[1], and Andreas Schaad[2]

[1] SAP Research, Karlsruhe, Germany
philip.robinson@sap.com
florian.kerschbaum@sap.com
[2] SAP Research, Sophia Antipolis, France
andreas.schaad@sap.com

Abstract. A choreography specifies the interactions between the resources of multiple collaborating parties at design time. The runtime management of authorization policies in order to support such a specification is however tedious for administrators to manually handle. By compiling the choreography into enhanced authorization policies, we are able to automatically derive the minimal authorizations required for collaboration, as well as enable and disable the authorizations in a just-in-time manner that matches the control flow described in the choreography. We have evaluated the advantage of this utility in a collaborative engineering scenario.

1 Introduction

We present a system architecture and algorithm for automatically generating, installing and enforcing authorization policies that correspond to an agreed specification of inter-organizational collaboration. The planning and execution of inter-organization collaboration is known as *business process choreography* [1,7,16], with the specification document being referred to as a *choreography description*. From a choreography description we were able to automatically derive the corresponding authorization policies. Moreover, authorizations (which are determined based on authorization policies) are enabled only for the duration of their corresponding interactions in the running collaborative business process. We ensure that an organization is quickly prepared to fulfill its obligations in a collaborative business process, while obeying the least privileges principle. In collaborative business there is a need to reduce both the risk of losing market credibility, due to slow response, and the risk of exposing valuable information. This challenge is presented in collaborative engineering, where multiple organizations collaborate on a short term basis within a specific project, in order to exchange design objectives, detect and resolve conflicts, as well as generate new ideas and design options [2]. In order to support collaborative business processes such as collaborative engineering, new methodologies, specification languages and tools [1,7,10,16] are being produced, alongside which we position and evaluate our work.

E. Damiani and P. Liu (Eds.): Data and Applications Security 2006, LNCS 4127, pp. 297–309, 2006.

The paper continues with a preliminary introduction to business process choreography, authorization and collaborative engineering, outlining the problem domain. Secondly, we present the system architecture, elements and component interactions for deriving authorization policies from a choreography description. This is followed by the details of the authorization policy generation algorithm, and a discussion of its merits. We conclude with a discussion of the solution and related work.

2 Preliminaries

2.1 Business Process Choreography

Business process choreography is the description of how multiple organizations coordinate their activities in a collaborative business process. A choreography may have been agreed to, but it is then the job of individual organizations to provide authorized access to the systems and resources that will do the actual specified work. The basic building blocks of a choreography are interactions, such as web service invocations across organizations, and internal actions within one organization. [16] explains this as a three stage process:

1. create a common understanding of the inter-organizational workflow (or collaborative business process) by specifying a shared public workflow,
2. partition the public workflow over the organizations involved, and
3. for each organization, create a private workflow which is a subclass of the respective part of the public workflow.

The choreography description therefore defines the public workflow, which acts as a means of combining the private workflows of multiple organizations into one global control-flow.

The Web Services Choreography Description Language (WS-CDL) [7] is an emerging, XML-based standard for describing message-based interactions between web services. We however only discuss the elements that were important for generating authorization policies. For a more complete introduction to WS-CDL see [1,7]. WS-CDL offers an element for encoding a web service call, referred to as an `<InterAction>` element. This contains references to the source `<roleType>` and the receiver `<roleType>`. In addition the expected internal actions of participant organizations are denoted using the `<SilentAction>` and `<NoAction>` tags. As control-flow elements the WS-CDL specification offers sequential `<Sequence>`, parallel `<Parallel>`, branching `<Choice>` and looping `<WorkUnit>` composition, which we take into consideration when specifying dependencies between generated authorization policies.

2.2 Authorization and Access Control

Having described and agreed to a choreography, each collaborating organization still needs to provide the appropriate authorizations that allow the agreed interactions to be executed. Furthermore, authorizations should only be activated

according to the control-flow of the choreography. An authorization is defined as a triple ⟨subject s, object o, action a⟩, stating that a subject s can perform action a on object o [4,12]. Messages originating from a subject and targeted at an object are composed of a corresponding triple. Access control is then the process of intercepting every incoming message before it reaches its target, and determining whether or not the request can be granted [12]. The decision requires policies to be specified that consider the message plus the conditions under which it was received. A generic authorization policy is then specified as $\langle s, o, a, q \rangle$, where q is a set of conditions that must evaluate to true in order for the $\langle s, o, a \rangle$ triple to be a valid authorization. A general architecture for access control therefore consists of a policy decision point (PDP), which makes the authorization decisions based on installed authorization policies, and a policy enforcement point (PEP), which intercepts all incoming messages and enforces the authorizations or denials resulting from policy decisions [12]. A message also consists of a triple $\langle s, o, a \rangle$, such that its authorization is evaluated according to a policy, whose $\langle s, o, a \rangle$ triple matches the corresponding elements of the message. In order for an authorization decision to be made, the PEP first authenticates the identity of s in all intercepted messages, then forwards to the PDP. The PDP then makes decisions concerning if $\langle s, o, a \rangle$ is valid given q, a set of condition variables.

3 Collaborative Engineering

Having discussed the background of our work (choreography and authorization), we now consider this in the context of a specific application domain. We have selected collaborative engineering, as the issues of managing short-term, inter-organization control flow arise, along with the requirement to provide minimal access to sensitive documents and services. Collaborative engineering is a way in which multi-functional development teams coordinate their communication and work in order to improve the process of developing a new product. It involves different partners with different perspectives on the engineering tasks [2]. We draw an example from the aerospace engineering domain, as depicted in figure 1, where they use grid and web-service technology in order to share resources [3]. The engineering team partners, computational resources and data are not part of the same administrative domain. Therefore each organization maintains and administers its own PDP, PEP and services it provides, as well as the Policy Generation Component (PGC) introduced in section 4. Each partner has a right to protect access to the resources they own, yet must still maintain their obligations to complete the business process. The team partners involved are discussed with respect to their authorization and collaboration requirements:

– *Initiator*, in this case the Aircraft Company, specifies the overall requirements for the product to be designed during the project. The series of design documents need to be version controlled and accessed only for specific project tasks. Leaking the design documents could destroy the opportunity to gain a market or patent. The Initiator adds and removes partners in the

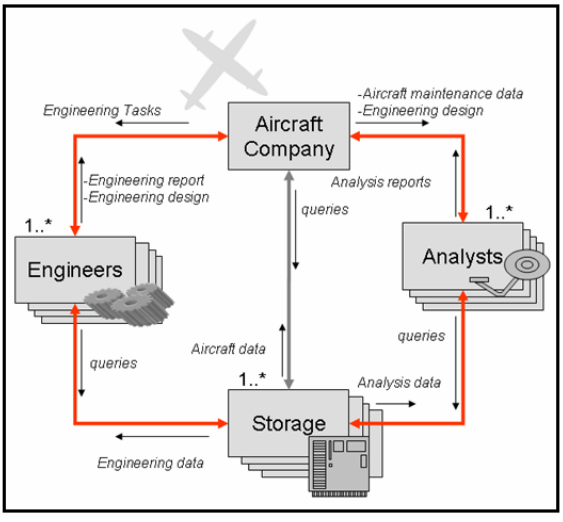

Fig. 1. Scenario for Collaborative Engineering in the Aerospace Domain

project based on their performance or changes in the environment (e.g. if a partner becomes a competitor).

- *Storage Providers* are contracted to store, version control and maintain access to large design documents, analysis reports and simulation data. A Storage Provider must ensure that access to shared resources does not violate the rules of the document owners nor the availability requirements of the contract.
- *Engineers* are contracted to provide models that meet the Initiator's requirements specification, using their own methods and tools. The Engineers may maintain models on their local machines or use the computational facilities of a Storage Provider.
- *Analysts* are contracted to provide models of the environment where the product is to be operational and therefore make predictions about how the final product will perform in a live environment. Their access is limited to very specific specification and design documents.

In order to coordinate the activities between the different specialist teams, a series of notifications and requests are interchanged, to which particular response actions occur. The notifications and requests are messages that state that an action is to be performed or an explicit attempt by a subject to perform an action on a resource such as read, write or delete. Coordination and authorization are critical throughout the lifetime of the project, with respect to confidentiality, availability and performance. For example, analysts should only be able to access design documents when a draft had been agreed in an earlier interaction, otherwise extra effort must be invested in organizational conflict analysis and resolution [2]. The partners may change during the lifetime of the project, such that the authorizations of old partners should be immediately removed. The

required guarantees are that resources are available when they need to be available and only to whom they need be available.

4 System Architecture

The PDP and PEP components form the basic trusted computing base of our system architecture, but we introduce an additional component for authorization policy generation called the PGC. In addition to generating authorization policies, the PGC is also responsible for installing them on the PDP. Each of these components are assumed to be trusted, as there is no intermediate policy decision that intercepts their interactions and they are assumed to be in the same administrative domain. We do not cover the administration authorization model of these components in this paper.

- Policy Generation Component (PGC): interprets a choreography and generates authorization policies. In addition to the choreography (WS-CDL), the service description (WSDL) containing the end point references (EPRs) of the target objects, as well as the public key certificates (PKCs) of the selected parties in the choreography are received.
- Policy Enforcement Point (PEP): intercepts requests to the resources and extracts authorization queries of the form $\langle s, o, a \rangle$, authenticates the subject of the message msg and queries a PDP for an authorization decision. Only authenticated and authorized messages msg_{auth} are allowed to reach targeted resources. Typical examples of where PEP functionality is implemented are network routers, switches, firewalls, proxies, OS filesystem interfaces and database interfaces.
- Policy Decision Point (PDP): makes authorization decisions based on $\langle s, o, a, q \rangle$ policies that have been generated by a PGC. A PDP receives a triple $\langle s, o, a \rangle$ from the PEP and outputs either an authorization msg_{grant} or denial message msg_{deny}.
- Resources: the objects to which access is requested, as agreed to in the choreography. We assume that there is a standard means of representing and exposing the interface to these resources as services, such as WSDL, but the issues of interoperability and interconnection are not discussed in the paper.

4.1 Component Interactions and Assumptions

Before describing the component interactions, there are some assumptions that we make with respect to a particular instance of the environment within which they interact. Firstly, we assume that the PEP, PDP and PGC are all in the same administrative domain as the resources being protected.

Each project partner's PEP is therefore situated in a logical DMZ (demilitarized zone), while the PDP, resources and PGC are in a protected domain. Secondly, we assume that there is a PKI (Public Key Infrastructure) in place that allows each project partner to validate the certificates of each other. The

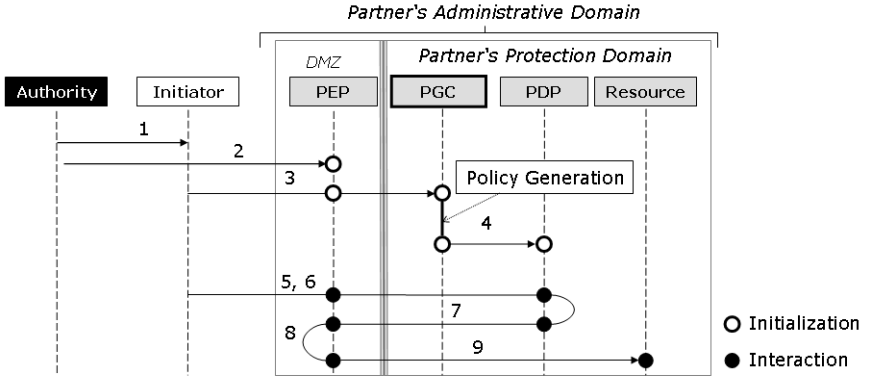

Fig. 2. Sequence diagram of component interactions

"Authority" in figure 2 represents a standard PKI Certificate Authority (CA). Finally we assume that each partner maintains a project management system that at least provides information concerning which projects are currently active. The component interactions are represented in figure 2 and discussed below:

1. The Initiator (i.e. Aircraft Company) must first be issued a public-key certificate (PKC) from an authority, asserting its identity and claim to be the initiator of a project with a unique universal identifier PRJ_{id}. The certificate has the format: $Cert(subject = Initiator,\ keyinfo = PK_{Initiator},\ extension = Claim(isInitiator, PRJ_{id}),\ issuer = PK_{Authority})$
2. The identities of all partners must also be validated by an authority, as well as the claim that they have been selected to play a role r_{name} in the project PRJ_{id}. $Cert(subject = Partner,\ keyinfo = PK_{Partner},\ extension = Claim(r_{name}, PRJ_{id}),\ \ldots,\ issuer = PK_{Authority})$
3. The Initiator then sends the WS-CDL (choreography), WSDLs (service interfaces) and PKCs of all selected partners to the PGC of each partner, using the initiator certificate as its authentication token. Recall that the PGC is not directly accessible from outside the network, such that there must be an authorization in place that allows $s = Initiator, o = PGC, a = Add$, under the conditions $q : isActive(Initiator, Partner, PRJ_{id})$.
4. The PGC derives the authorization policies and installs them on the PDP, given that the above condition q holds. This allows administration to simultaneously confirm participation in the collaborative business process and determine valid authorizations.
5. The Initiator issues an initiation message to all partners, indicating that the project is in the operation state. A PGC of partners implementing our proposed architecture must simply enable the first authorizations according to the choreography.
6. Incoming message requests are intercepted by each partner's PEP. We assume that there is mutual authentication between the communicating

parties. If the authentication is successful, the PEP extracts the authorization-relevant information ($\langle s, o, a \rangle$) and uses this to query the PDP.
7. The PDP evaluates the request by the PEP and returns its decision (msg_{grant} or msg_{deny}). Every time a policy decision is made, the PDP updates its internal state with the next set of authorizations to be installed.
8. The PEP drops all msg_{deny} and forwards all msg_{grant} to the appropriate resource, e.g. analysis service.
9. The actual resource is then invoked using only authorized messages (requests and invocations) forwarded by the PEP.

We now proceed to describe the algorithm for implementing the policy derivation.

5 Control-Flow Aware Authorization Policy Derivation

Our policy derivation algorithm uses the control flow of the choreography in order to minimize the time a policy is enabled. In addition to extracting only the relevant `<InterAction>` elements and enabling them over the life-time of the choreography, we also use the control-flow of the choreography to trigger the enabling and disabling of the policies, given the following extensions to the basic specification.

5.1 Extended Authorization Policies

We extended the specification of authorization policies in order to develop a mechanism for supporting the control-flow of a choreography. Instead of representing the run-time control-flow (and monitoring it with a second component) we have chosen to represent the static control-flow and extend the policy enforcement and decision with two mechanisms to track the control-flow. Each authorization policy is annotated with two additional fields, one set of policies that are enabled and one with policies that are disabled after the policy has been successfully matched. Let $l_{enable} = \{policy - id_1, policy - id_2, \ldots, policy - id_n\}$ be the set of policies to enable and $l_{disable} = \{policy - id_1, policy - id_2, \ldots, policy - id_m\}$ be the set of policies to disable. Additionally we store the $policy - id$ of each policy as an unique identifying integer and the state of the policy. The state of a policy can be either *enabled* or *disabled*. Disabled policies are not considered when making a policy decision, but they have already been created and can be activated on-demand, i.e. $q : enabled(policy - id)$. This allows us to create all policies initially and then reference them by policy-id for enabling and disabling. The life-cycle of a policy is depicted in figure 3. Furthermore, a policy might be enabled and disabled multiple times during the execution of a choreography. Summarizing a policy is a 7-tuple of $\langle policy - id, s, o, a, l_{enable}, l_{disable}, state \rangle$ where s, o and a are the usual authorization policy elements mentioned above.

The sets l_{enable} and $l_{disable}$ are evaluated by the PDP after a policy has been successfully matched. I.e. the PDP evaluates all enabled policies in order, comparing them to the request, and, on the first policy matching subject s, object o

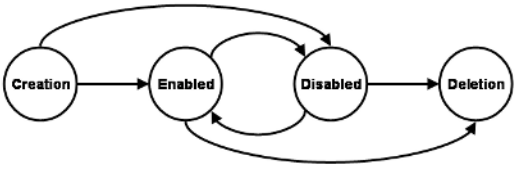

Fig. 3. Policy State Model

and action a it allows access. All our policies are *grant* policies specifying an allowed access. The PDP has an implicit *deny* policy that is used for all unmatched requests. Since we only have *grant* policies there is no conflict specified by our policies. Furthermore, if all policies are generated by the policy generator they are non-overlapping. After a policy has been successfully matched to the request the PDP processes the set l_{enable} of this policy and enables all policies in the set, followed by the set $l_{disable}$ of this policy disabling all policies in the set. This implies that a policy might disable itself after successful matching. Our algorithm ensures that no policy appears in both sets (l_{enable} and $l_{disable}$), such that the order of evaluation causes no conflict.

5.2 Automatic Policy Generation

In this section we will describe the algorithm to populate the enable and disable sets of each policy. WS-CDL offers four activities for control-flow: `<Sequence>`, `<Choice>`, `<Parallel>` and `<WorkUnit>`. `<Sequence>`, `<Choice>` and `<Parallel>` are self-explaining and work like expected. It is important to note that `<Parallel>` introduces parallelism into the control-flow which complicates the analysis. `<WorkUnit>` encapsulates another activity and makes its execution conditional. The enclosed activity may be executed 0 to n times depending on the guard and repetition conditions.

As we are interested in deriving authorization policies, the only activity that results in a (cross-domain) access to a resource is `<InterAction>`. Each `<InterAction>` represents an access from one party to another (i.e. no multi-party access).

We first present an overview of our algorithm and then detail its important steps. The algorithm steps are as follows:

1. Verify all partner certificates carry the role extension with the choreography role name. We place the canonical name of the subject of the certificate as the subject s into the authorization policy. The PDP can then match the subject of the certificate presented for authentication to the subject s in the authorization policy.
2. Derive control-flow with all intermediate nodes (i.e. XML elements, like `<Choice>` and `<Parallel>`) from the parsed choreography description.
3. Filter all `<InterAction>` elements that do not have oneself as a target, i.e. remove all `<InterAction>` elements that do not result in a local access (and therefore a local authorization policy).

4. Remove all XML nodes that do not contain `<InterAction>` elements by directly linking predecessors and successors. (Exception: Empty nodes in `<Parallel>` are simply removed.)
5. Create policies with empty l_{enable} and $l_{disable}$ sets in disabled state.
6. Compute l_{enable} and $l_{disable}$ set for each policy from the choreography.
7. Install policies on a partner's PDP and enable start policy (or policies) on request.

The derivation of the control-flow is straightforward from the syntactic constructs of WS-CDL. We store the predecessor and successor information in a two-dimensional array in the activity. Let $pred_i(N)$ denote the set of sequential predecessors of node N for $i = 1, \ldots, p$ where i is the i-th parallel activity, i.e. one of each $pred_0(N)$, $pred_1(N)$, ..., $pred_p(N)$ has occurred before N. Similarly let $succ_i(N)$ denote the set of sequential successors for the i-th parallel activity, i.e. one of each $succ_0(N)$, $succ_1(N)$, ..., $succ_q(N)$ will occur after N. Let $pred(N)$ and $succ(N)$ denote the union of all parallel activities, i.e. $pred(N) = \bigcup_{i=0}^{p} pred_i(N)$ and $succ(N) = \bigcup_{i=0}^{q} succ_i(N)$. Let $pred_N(M)$ and $succ_N(M)$ denote the set of predecessors and successors, respectively, of M of that activity that contains N. Due to the syntax of `<Parallel>` and `<Choice>` in WS-CDL this set is unique. We use the same notation when applying predecessor and successor operators to sets, i.e. the operator is applied to each element in the set removing duplicate results. E.g. $succ(pred(N))$ denotes the set of all possible sequential and parallel siblings of N (including N itself).

The creation of policies in step 5 requires the subject, object and action information for each policy. We analyze each `<InterAction>` for this purpose. The subject is the canonical name as extracted from the certificate for the role (attribute `fromRole` in the `<participate>` element). The object is the web service EPR from the WSDL file referenced in the `<behavior>` element of the `<roleType>` with the name of the `operation` attribute of the `<InterAction>`. The action is the method of the web service that is called and, in our convention, is referred to as the `operation` attribute. With this information we construct a draft of each policy, such that all policies are disabled, but have a policy-id assigned. Each such policy is associated with a node in the control-flow and, after the removal of empty nodes, each node has one policy associated with it. I.e. we can use control-flow nodes and policies interchangeably.

The set $l_{disable}$ is the set of all alternative sequential choices, i.e. for each node N

$$l_{disable} = succ_N(pred(N))$$

Note that this includes N itself, i.e. by default each policy is disabled after it has been activated unless the control-flow allows it to be reactivated.

The set l_{enable} is the set of all successors of a node N

$$l_{enable} = succ(N)$$

Since a node may be contained in both $l_{disable}$ and l_{enable}, e.g. a loop to itself (in a `<WorkUnit>` activity with repetition), we postprocess both sets and remove all elements that are contained in both sets from the sets.

$$\phi = l_{disable} \cap l_{enable}$$

$$l_{disable} = l_{disable} \setminus \phi$$

$$l_{enable} = l_{enable} \setminus \phi$$

The policies are now ready for deployment and we only need to enable the start policies to allow the choreography to start.

6 Discussion and Related Work

An authorization is said to be passive with respect to a given condition, if for all possible values of the condition's variables the authorization remains unchanged. Conditions include tasks performed, time of day, resource availability and various other properties that can be used to describe a system's current operational state and behavior. The implications of "active", "just-in-time" or "need-to-know" authorizations have been discussed by various authors including [5,15,17]. We consider the authorizations generated from a choreography as active with respect to project membership, role, task and control flow. These therefore maintain the membership of a group and enforce access controls on resources and functions that are reserved for active and qualified partners of the group. When an organization needs to enforce an access control on a resource reserved for usage in a project, it is therefore important to have a means of validating the membership of the subject with the project, otherwise residual access could be granted to a requesting subject that is no longer an active partner.

6.1 Active, Task and Membership-Based Access Control

TBAC [15] has similar foundations as RBAC [11], with the goal of modelling authorizations at the enterprise and application level as opposed to restricting them to the system and resource level. TBAC authorizations are granted and revoked based on when tasks are scheduled and performed, such that permissions to objects should be granted to subjects only for the duration of a task that necessitates the subject performing some action on the object. TBAC's authorization policy extensions are two fields for activating or deactivating authorizations at runtime. A TBAC authorization has the form $\langle s, o, a$, usage u, authorization-step $as\rangle$. Each task in a workflow is associated with an authorization-step, representing the workflow's protection state when the task or activity is being performed. An authorization such as $\langle s, o, a\rangle$ is only valid when contained in the current authorization-step. Secondly, authorizations are conditioned by a usage or validity count u, specifying the number of times the authorization can be granted in the workflow. The authorization-steps are similar to our l_{enable} and $l_{disable}$ sets, but seem to assume that parallel activities will conclude at the same time. Our approach therefore offers a finer granularity with the enabling and disabling of authorizations. Secondly, TBAC must be integrated with a workflow engine in order to function, creating a large trusted computing base beyond standard PEPs and PDPs.

TMAC (Team-based Access Control) [14] is a related framework to TBAC and RBAC, but adds the feature of authorization enabling based on team existence and membership. That is, a subject s may be assigned a role r_t containing a set of permissions $\langle o, a \rangle$, but these roles are only active when a relevant team T exists and s plays role r_t in T. The authorizations are deactivated either by removing the role assignment or the team assignment. A project is a team of multiple participants, such that we achieve a similar membership-based access control activation and deactivation. We however support a finer-grained activation and deactivation mechanism, such that we achieve dynamic authorization based on role, task and membership.

6.2 Generation of Access Controls

[8] also addresses the idea of adding workflow states to the protection state variables of a system. They use Petri nets as the basis for modeling workflows, as the theoretical and practical understanding of Petri nets to modeling information and control flow are well established. The activities of a workflow correspond to the transitions in a Petri net, while the workflow state, data-stores and control flow are represented by the places and markings. Their extensions to the authorization policies are quite simple, as they express an authorization as $\langle s, o, a, task \rangle$, stating that a subject s is allowed to perform action a on object o if $q : isScheduled(task)$. They also define a simple read/write $\{r, w\}$ policy that is associated with incoming and outgoing data of a task. That is, if a subject s is assigned to a task $task_i$ with incoming data o_i and outgoing data o_j, then s should have read 'r' rights to the data-store of o_i and write 'w' rights to the data-store of o_j. The claim of their work is that given the appropriate Petri net model of a workflow, by applying the $\langle s, o, a, task \rangle$ scheme and the $\{r, w\}$ policy the required authorizations can be derived and enforced at runtime of the workflow. They envisioned the enforcement being done by a workflow engine as tasks were scheduled. Again this is tight coupling of functionality and too large a trusted computing base. A workflow engine is already a complicated piece of software that must maintain state, manage concurrencies, handle exceptions and perform compensation actions. We believe that their simple derivation scheme could work, but suggest decoupling authorization and workflow management. [6] achieve this decoupling by using task-based capabilities, which are assigned to subjects that have specific roles in the workflow. A PDP maintains a matrix of tasks and objects, where the cells specify actions that holders of task-based capabilities are allowed to perform on these objects. Our architecture and algorithm manage to capture a combination of these concepts.

[9] have taken advantage of the existence of maturing standards for defining business processes and workflows, such as the Business Process Execution Language for web services (BPEL4WS). They have presented a conceptual integration of BPEL with RBAC in order to provide an authorization concept to accompany BPEL. They have defined a conceptual mapping between their interpretations of the meta-models of BPEL and RBAC. While what we achieve is also a transformation or mapping, this is still only a first step when considering

the requirements of collaborative engineering projects that need to meet tough deadlines. The enabling, disabling and removal of these derived authorizations must be supported with respect to the control flow of the process from which they have been derived.

7 Conclusions

We have described an architecture and algorithm for deriving authorization policies from a business process choreography. This enables partners in e.g. a collaborative engineering project to focus on agreeing on the protocol for how they collaborate and still have the assurance that their authorization requirements will be addressed. Our approach has been implemented and is being applied in the context of a larger research project – TrustCoM [13] – for security in collaborative business processes, which includes collaborative engineering as a main test case, and also considers legal and business elements that influence the validity and control flow of collaboration. Future work will therefore consider how these can be included into our framework, as well as handling faults and exceptions.

Acknowledgements

The developments presented in this paper were partly funded by the European Commission through the IST programme under Framework 6 grant 001945 to the Trustcom Integrated Project.

References

1. A. Barros, M. Dumas, and P. Oaks. A Criticial Overview of the Web Services Choreography Description Language. BPTrends, 2005.
2. M.R. Danesh and Y. Jin. An Aggregated Value Model for Collaborative Engineering Decisions, Proceedings of the 5th ASME Design for Manufacturing Conference, 2000.
3. A. Gould, S. Barker, E. Carver, D. Golby, M. Turner, BAEgrid: From e-Science to e-Engineering, in Proceedings of the UK e-Science All Hands Meeting, 2003.
4. M. Harrison, W. Ruzzo, and J. Ullman. Protection in Operating Systems. Communications of the ACM 19(8), 1976.
5. R. Holbein, S. Teufel, and K. Bauknecht. The use of business process models for security design in organisations. In Proceedings of SEC, 1996.
6. M. Kang, J. Park, and J. Froscher, 2001. Access control mechanisms for inter-organizational workflow. In Proceedings of the 6th ACM Symposium on Access Control Models and Technologies, 2001.
7. N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web Services Choreography Description Language Version 1.0. Available at *http://www.w3.org/TR/ws-cdl-10/*, 2005.
8. K. Knorr. Dynamic access control through Petri net workflows. In Proceedings of the 16th Annual Computer Security Applications Conference, 2000.

9. J. Mendling, M. Strembeck, G. Stermsek, and G. Neumann. An Approach to Extract RBAC Models from BPEL4WS Processes. In Proceedings of the 13th IEEE international Workshops on Enabling Technologies, 2004.
10. P. Robinson, Y. Karabulut, and J. Haller. Dynamic Virtual Organization Management for Service Oriented Enterprise Applications. In Proceedings of IEEE CollaborateCom, 2005.
11. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. IEEE Computer 29(2), 1996.
12. P. Samarati, S. de Capitani di Vimercati. Access Control: Policies, Models, and Mechanisms. Lecture Notes in Computer Science 2171, 2001.
13. T. Dimitrakos, S. Ristol, and M. Wilson. TrustCoM: A Trust and Contract Management Framework for Dymamic Virtual Organisations. ERCIM News Magazine, 2004.
14. R. Thomas. Team-Based Access Control (TMAC): A Primitive for Applying Role-Based Access Controls in Collaborative Environments. In Proceedings of the 2nd ACM workshop on Role-basedAccess Control, 1997
15. R. Thomas, and R. Sandhu. Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Autorization Management. In Proceedings of the IFIP 11th International Conference on Database Securty, 1998.
16. Wil M.P. van der Aalst, Mathias Weske. The P2P Approach to Interorganizational Workflows, Lecture Notes in Computer Science, 2001
17. W. Yao, K. Moody, and J. Bacon. A Model of OASIS Role-Based Access Control and its Support for Active Security. Proceedings of 6th ACM Symposium on Access Control Models and Technologies, 2001.