Coordination based Distributed Authorization for Business Processes in Service Oriented Architectures

Sarath Indrakanti Vijay Varadharajan
Information and Networked Systems Security Research
Dept. of Computing, Macquarie University, Australia
{sindraka, vijay}@ics.mq.edu.au

Abstract — **Design and management of authorization services in service oriented architectures poses several challenges. In this paper, we propose authorization architecture for business process layer in service oriented architecture. We describe the components and functionalities of the architecture such as authorization policy evaluators, certificate and credential authorities and dynamic attribute services and discuss the security management of these functions at specification time and at run time. Then the paper describes authorization evaluation algorithms and discusses the design choices for evaluation models. Finally, the paper describes the benefits of the proposed architecture, which has been implemented.**

Keywords- Authorization, Business Processes, Service Oriented Architectures

I. INTRODUCTION

Broadly speaking, the Service Oriented Architecture (SOA) comprises web services and business workflows built using web services. These workflows are called business processes [1]. Figure 1 shows the positioning of the authorization service components within the various layers of the SOA. Authorization services for the web services layer have special design requirements because web services present a complex layered system. For instance, a service could be a front-end to an enterprise system where the enterprise system accesses information stored in databases and files. Web services may be used by enterprises to expose the functionality of legacy applications to users in a heterogeneous environment. Alternatively, new business applications could be written to leverage benefits offered by web services. This means that authorization architecture for web services must support multiple models of access control. This enables legacy applications to use the access control models they have already been using as well as new web services applications to use new models of access control.

Currently, there exist a range of authorization models for stand-alone systems and traditional distributed systems. There also exist a few authorization schemes that are designed either for the web services layer [2,3] or the business process layer [4, 5] of the SOA. There is no unified model currently available that provides a comprehensive authorization framework for both web services and business processes comprising the SOA. After carrying out a thorough survey and analysis of the existing authorization models built for stand-alone systems and traditional distributed systems as well as for various layers of the SOA, we have formulated the design requirements for authorization services required for web service and business process layers. These are described in Section II.

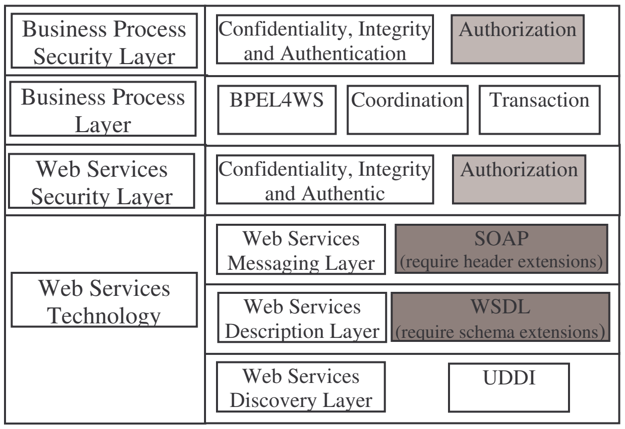
Business Process Security Layer
Confidentiality, Integrity and Authentication
Authorization
Business Process Layer
BPEL4WS
Coordination
Transaction
Web Services Security Layer
Confidentiality, Integrity and Authentic
Authorization
Web Services Technology
Web Services Messaging Layer
SOAP (require header extensions)
Web Services Description Layer
WSDL (require schema extensions)
Web Services Discovery Layer
UDDI

Fig. 1: Layers in the SOA

Taking into account these design principles, we have proposed a unified authorization framework for the SOA. The authorization framework comprises two separate authorization architectures (indicated by the light-grey coloured boxes in Figure 1 that extend the security layers of web services and business processes. Extensions to the web services description and messaging layers are also proposed to support the unified authorization framework for the SOA (indicated by the dark-grey coloured boxes in Figure 1). This work builds on the work on the authorization architecture for web services, referred to as the Web Services Authorization Architecture (WSAA) [6]. In this paper, our focus is on the business process layer authorization architecture, which is referred to as the Business Process Authorization Architecture (BPAA). The BPAA builds on top of the WSAA and forms part of the overall authorization framework for Service Oriented Architectures. BPAA is the focus of this paper.

Authorization architecture for the business process layer of the SOA must provide orchestration services to coordinate the authorization decisions from individual partner's authorization policy evaluators. Each partner must be allowed to control its own authorization policies and also not require disclosing them to the entire workflow or to the workflow engine. Even in cases where the binding to actual end-points of partner services happens dynamically at runtime, the authorization architecture must be able to orchestrate the partners' authorization policy evaluators and arrive at an authorization decision.

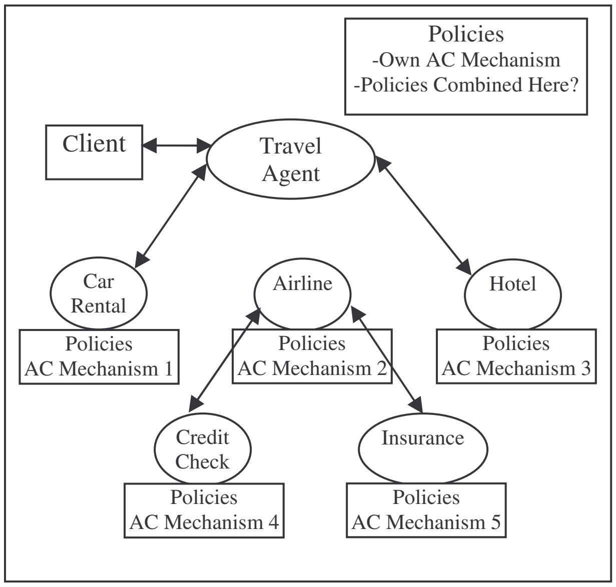

Fig. 2: Travel Agent Service Example

Consider for instance the Travel Agent Service shown in Figure 2, where each of the partner services may potentially use their own access control (AC) mechanisms. The partner airline (for example, United Airlines) may not wish to disclose its policies to the travel agent. Similarly, other partners also may not wish to disclose their policies to be combined by the travel agent in order to authorize the client. In the course of the workflow, the client needs to get authorized seamlessly to partner services.

The paper is organized as follows. In Section II, we consider the requirements for authorization framework for Service Oriented Architectures. Section III describes the proposed business process authorization architecture. Finally Section IV concludes.

II. AUTHORIZATION DESIGN REQUIREMENTS FOR SOA

In the next two sub-sections, we outline the principles involved in the design of authorization framework for web services and business processes layers of the SOA.

A. Authorization Principles for Web Services Layer

(i) Support for multiple access control models — Authorization service must be able to support a range of access control models. This is necessary because it is not realistic to expect every web service-based application to use the same access control model. In fact, where web services are used to expose the functionality of legacy enterprise applications, it is likely that organizations will prefer to use their currently existing access control models and mechanisms that they have been using, before exposing the legacy applications as web services. Therefore, authorization architecture must be flexible enough to support multiple access control models including the traditional Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role Based Access Control (RBAC) and the Capability/Certificate-based Access Control models [7, 8].

(ii) Authorization Policies — Languages have long been recognized in computing as the ideal vehicles for dealing with the expression and the structuring of complex and dynamic relationships. Over the recent years, a language-based approach to specifying access control policies has (rightly) gained prominence; this is helpful for not only supporting a range of access control policies but also in separating the policy representation from policy enforcement. Hence an important design principle is to enable the support for a range of policy languages for specifying authorization policies. The policy language(s) used may support fine-grained and/or coarse grained authorization policies depending on the organization's requirements.

(iii) Authorization Credentials -- It is necessary to define what access-control related credentials are required and how to collect them. Some access control mechanisms may pull the credentials from the respective authorities and send them to the responsible authorization components. For example, in the semantic approach [9], the AC Proxy component collects the relevant privilege (attribute) certificates (for the client) from the PMI Client component which in turn requests the appropriate PMI Node for the privilege certificates for the client. Other access control mechanisms may expect the client to collect the credentials from the respective authorities and push them to the responsible authorization components. For example, [2] proposes a model in which a client itself collects the required authorization credentials from the relevant authorities and sends the set of credentials collected before invoking a web service. Hence, we recommend an authorization architecture designed for web services to be able to support both the push and pull models of collecting credentials.

(iv) Decentralized and Distributed Architecture — Given the distributed decentralized nature of the web, it is reasonable to demand that an authorization architecture designed for web services should embrace the same decentralized nature. As an example, an organization may typically have a hierarchical internal structure. The decentralized approach allows us to specify authorization policies for web services on an organizational unit level for different components in the web service hierarchy. A distributed architecture provides many advantages such as fault tolerance and better scalability, and can outweighs its disadvantages such as more complexity and communication overhead.

The WSAA architecture described in [6] has taken these principles into account in its design and management.

B. Authorization Principles for Business Process Layer

(i) Decentralized Policy Administration — Each partner involved in a business process workflow should be allowed to control its set of authorization policies autonomously whether the partners are from within one organization or from multiple organizations. The authorization architecture must not place a constraint on coordinating the policies across different domains involved in the virtual enterprise [4] formed by a business process workflow.

(ii) Dynamic Discovery of Business Process Partners' Authorization Evaluation Components — Every web service must let its potential clients be aware of the access control mechanism it uses and where, if necessary, to get the credentials from and where to send them for making the authorization decision. This could be achieved in the form of the assertions in the WS-AuthorizationPolicy (defined in [6]), similar to the WS-SecurityPolicy specification) statements. In the case of a dynamic business process, where the binding to actual implementations of partner web services is made at runtime using some pre-established criteria, the authorization architecture must make the client aware of each of the partners' authorization mechanisms and components involved. A coordination component may be used to send such information to clients. When the flow reaches a stage where some credentials are required by the access control system of the partner involved, the coordination component can make the client aware of what authorization credentials to send to the partner's component for the authorization evaluation to be made.

(iii) Orchestration of Partners' Authorization Evaluation Components and Combination of Individual Decisions — Authorization architecture must use some form of coordination mechanism to orchestrate the partners' authorization evaluation components and the client involved in a workflow. In the case of dynamic[1] business processes, a coordinator, for instance, should maintain session state information so that all or some partners know the authorization given to a client. As in any complex transactions handling mechanism, there must be a mechanism in place to either commit or rollback an authorization decision based on the authorization decisions from partners' policy evaluators. A business process may be performed only when all the partners' authorization components involved give out a positive authorization. Decision-combination algorithms such as those defined in the RAD architecture [10] should be defined by the partner controlling the workflow to combine and give out a final authorization decision.

[1] In a dynamic business process, only the partner interfaces are defined at the design time, but not the actual bindings to real instances of partner services.

(iv) Non-disclosure of Policies — A partner or a set of partners involved in a business process may not wish to disclose their policies to the partner that is controlling the business process. It is an important requirement that the authorization architecture should not need all the partners to disclose their policies to other partners involved in the workflow. For example, if a Travel Agent Service (TAS) creates a business process that binds and interacts with United Airlines, Hertz car rental and Hilton Hotel at design or runtime, the TAS should not require the different partners involved to disclose their policies to manage the authorization decisions involved. Large organizations would want to set and enforce their policies themselves or by outsourcing to a trusted partner (who runs their authorization service). However, they would want to do business by binding to portal travel agents using a secure SOA.

III. Business Process Authorization Architecture

Before we delve into the design of the architecture, we clearly distinguish between *static* and *dynamic* business processes. A static business process is a pre-composed business process, where all the partner service interfaces and their binding information are known at design time itself. A dynamic business process is more complex, where only the partner interfaces are defined at design time, but not the actual bindings to real instances of partner services (web services and/or business processes). The binding is made at runtime to real instances of services by letting the client interact with the business process. For instance, a travel agent may statically bind at design time to always book (i) flight tickets with United Airlines, (ii) cars with Hertz car rental and (iii) hotel rooms at the Hilton. But in real-world situations, customers want more flexibility; and therefore, travel agents may opt to expose their services as dynamic business processes, where the customer at runtime chooses an appropriate partner service (such as airline, car rental agency, or hotel) depending on their own requirements. We make an important assumption in this paper. A dynamic business process may not only invoke partner web services but also partner services that are themselves business processes.

A. BPAA Architecture Design

The proposed architecture is shown in Figure 3. The BPAA comprises an administrative domain and a runtime domain. We manage business processes in the administration domain. Authorization related components such as authorization policy evaluators, certificate and credential authorities and dynamic attribute services can be managed in the administration domain. Also security administrators can assign a set of authorization policy evaluators to authorize requests to business processes. We have a runtime domain where the authorization related information such as what credentials are required to invoke a particular business process and how to collect those credentials is compiled and stored. This makes the authorization process efficient. This information is automatically compiled from time to time when necessary by using the information from the administration domain and it can be readily used by components in the runtime domain. A client makes use of a registry server such as a UDDI directory to find business process definitions (WS-BPEL statements).

Let us now consider the various system components involved in the BPAA. The components *Authorization Policy Evaluators*, *Certificate and Credential Authorities*, *Dynamic Attribute Services*, and *Authorization Decision Composers* are system objects in our architecture. The *Authorization Manager (AZM)* for an organization is responsible for managing these components. The Authorization Administration API is used to manage these components and the related data is stored in the *Authorization Administration Database (AAD).*

The *Certificate and Credential Authority (CCA)* is responsible for providing authentication certificates and/or authorization credentials required to authenticate and/or authorize a client. For example, a CCA may provide authentication certificates such as X.509 or authorization credentials such as a Role Membership Certificate (RMC) or a Privilege Attribute Certificate (PAC). We define Certificate and Credential Authority as a tuple, *cca* = [*i, l, CR, pa, ra*(*pa*)], where *i* is a URN, *l* is a string over an alphabet Σ* representing a network location such as a URL, *CR* is the set of authentication certificates and/or authorization credentials *cca* provides, *pa* is an input parameter representing a subject, *ra* uses *pa* and gives out an output (result) that is the set of certificates/credentials for the subject.

The *Dynamic Attribute Service (DAS)* provides system and/or network attributes such as bandwidth usage and time of the day. A dynamic attribute may also express properties of a subject that are not administered by security administrators. For example, nurses may only access a patient's record if they are located within the hospital's boundary. A DAS may provide the nurse's 'location status' attribute at the time of access control. Dynamic attributes' values change more frequently than traditional *static* authorization credentials (also called privilege attributes). Unlike authorization credentials, dynamic attributes must be obtained at the time an access decision is required and their values may change within a session.

We define Dynamic Attribute Service as a tuple, *das* = [*i, l, AT, pd, rd*(*pd*)], where *i* is a URN, *l* is a string over an alphabet Σ * representing a network location such as a URL, *AT* is the set of attributes that *das* provides, *pd* is input parameter(s) representing attribute(s) name(s), *rd* uses *pd* and gives out an output (result) that is the value of the attribute(s).

The *Authorization Policy Evaluator (APE)* is responsible for making authorization decision on one or more abstract system operations. An APE may use a type of access control mechanism and an authorization policy language that may be unique to it. However, we define a standard interface for the set of input parameters an APE expects (such as subject identification, object information and the authorization credentials) and the output authorization result it provides.

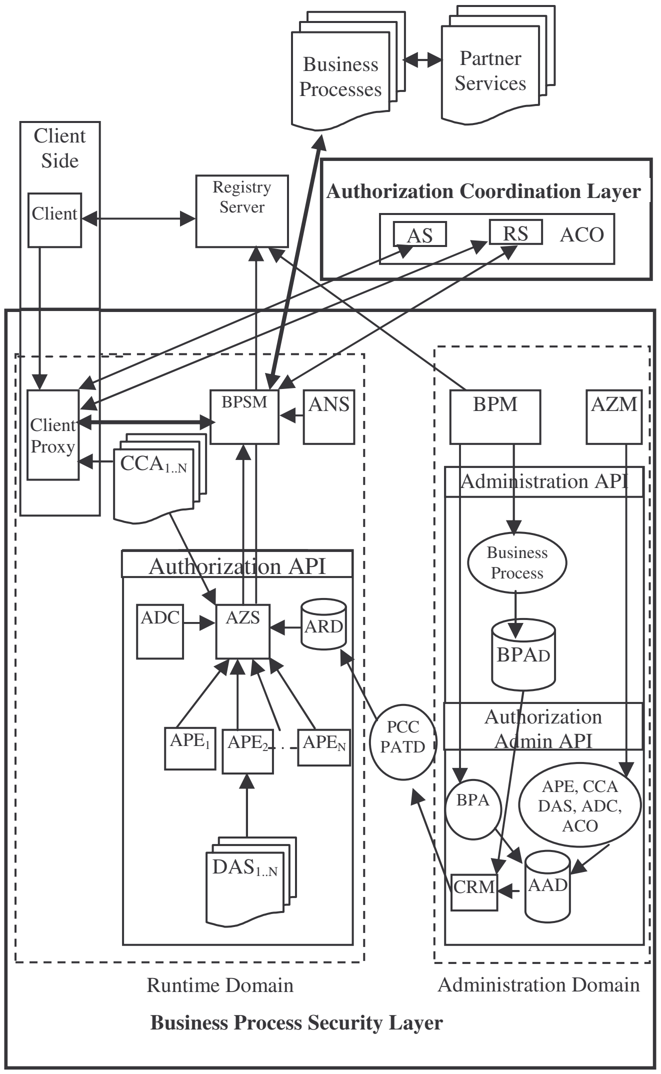

Fig. 3: BPAA Overview Diagram

We define Authorization Policy Evaluator as a tuple, *ape* = [*i, l, pe, re(pe), OP, DAS, CCA*], where *i* is a URN, *l* is a string over an alphabet Σ * representing a network location such as a URL, *pe* is the set of input parameters such as subject and object details, *re* is a function that uses *pe* and gives out an output (result) of authorization decision. *OP* is the set of abstract system operations for which *ape* is responsible. The *DAS* is the set of dynamic attribute services responsible for providing dynamic runtime attributes to the *ape*. The *ape* uses these attributes to make authorization decisions. The *CCA* is the set of certificate and credential authorities that provide the credentials required by the *ape*.

The *Authorization Decision Composer (ADC*) combines the authorization decisions from various authorization policy evaluators involved by using an algorithm that resolves the authorization decision conflicts and combines them into a final decision. We define Authorization Decision Composer as a tuple, *adc* = [*i, l, a, pc, rc(pc)*], where *i* is a URN, *l* is a string over an alphabet Σ * representing a network location such as a URL, *a* is the name of a pre-defined algorithm *adc* uses to combine the decisions from the individual authorization policy evaluators. The *pc* is an input parameter representing the decisions from individual authorization policy evaluators, *rc* uses *pc* and the authorization decision composer algorithm *a* to combine the decisions and gives out an output (result) that is the value of the final authorization decision.

The runtime domain consists of the Client Proxy, the Business Process Security Manager, the Authentication Server, the Authorization Server, and the Authorization Coordinator components.

The *Client Proxy (CP)* collects the required authentication certificates and/or the authorization credentials from the respective authorities on behalf of the client before sending a request to a business process and handles the session on behalf of the client with a *Business Process Security Manager*.

The *Business Process Security Manager (BPSM)* is responsible for both the authentication and the authorization of the client to a business process. A client's Client Proxy sends the necessary authentication certificates and authorization credentials to the BPSM. It is responsible for managing all the interactions with a client's Client Proxy.

The *Authentication Server (ANS)* receives the authentication certificates from the BPSM and uses a mechanism to authenticate the client. We treat the ANS as a black box in our architecture as our focus in this paper is on the authorization of the client. We included this component in the business process security layer for completeness.

The *Authorization Server (AZS)* decouples the authorization logic from the application logic. It is responsible for locating the business process' Authorization Policy Evaluators, sending the credentials to them and receiving the authorization decisions. Once all the decisions come back, it uses the business process' Authorization Decision Composer to combine the authorization decisions. If required, the AZS also collects the required authorization credentials on behalf of clients from the respective Certificate and Credential Authorities.

The *Authorization Coordinator (ACO)* is used to coordinate the authorization between a client (by involving the Client Proxy) and the *dynamic* business processes and their partner services (web services and/or business processes). It is composed of an *Activation Service (AS)* and a *Registration Service (RS)* that expose standard interfaces to the participants (Client Proxy and Business Process Security Manager) in the authorization coordination protocol.

The *Business Process Managers* (BPMs) manage a set of business processes for which they are responsible in an organization. They use the Administration API shown in Figure 3 to manage the business processes. The business process definitions are stored in the *Business Process Administration Database* (BPAD), see Figure 3.

We define a Business Process as a tuple, $bp = [i, l, \Sigma, WS, BP, B, pa, MD, bpm, bpsm, aco]$, where *i* is a non-empty string over an alphabet Σ^* representing a globally unique identifier such as a URN, *l* is a string over an alphabet Σ^* representing a network location such as a URL, Σ is a finite set of states representing the internal state of the business process at a given time, *WS* is the set of URNs of partner web services or activities that comprise the business process, *BP* is the set of URNs of partner business processes or activities that comprise the business process, *B* is the network protocol binding such as SOAP over HTTP for the business process, *pa* represents the business process flow algorithm represented in a WS-BPEL statement, *MD* is the metadata providing additional description for *bp*, the *bpm* is the identity (ID) of the Business Process Manager (BPM) responsible for managing *bp*. The *bpsm* is the location of the Business Process Security Manager component responsible for the authentication and authorization of the clients to the business process. The *aco* is the location of the Authorization Coordinator responsible for coordinating the authorization of a client to the *bp*'s partner services. The *aco* is defined only for dynamic business processes and is null for static business processes.

The Σ, *B*, or the *MD* can be the empty set Ø. If *B* is an empty set, Ø, then the business process defined is either an *abstract* business process or a dynamic business process. An abstract business process is not executable and only defines the standard interfaces between a business process and its partner services and the messages passed between them. If it is a dynamic business process, the individual bindings to partners are made at runtime using the client's preferences. If *B* is not an empty set at business process design time, then it is a static (pre-composed) business process.

B. BPAA Authorization Policy Evaluation

The Business Process Managers (BPMs) are also responsible for managing the authorization-related information for the business processes for which they are responsible. This information is stored in the *Business Process Authorization* tuple, $bpa = [i, bp, APE_{bp}, adc_{bp}]$, where *i* is a URN, *bp* is the business process to which *bpa* is defined. The APE_{bp} is the URNs of the set of Authorization Policy Evaluators responsible for authorizing the requests from a client to the *bp*. The adc_{bp} is the URN of an *Authorization Decision Composer*. It is responsible to combine at runtime, the authorization decisions given out by the set of APEs in the APE_{bp}.

At the time of evaluation in runtime domain, Credential Manager (CRM) component in the BPAA is responsible for compiling and storing the authorization information required by the components. This runtime authorization information is stored in the *Authorization Runtime Database* (ARD) (Fig. 3). The runtime authorization information consists of two tuples namely, BusinessProcess-Credential-CCA tuple (PCC tuple) and BusinessProcess-Attribute-DAS tuple (PATD tuple).The CRM is invoked from time to time, when a business process object is created or modified in the BPAD.

The BusinessProcess-Credential-CCA tuple is defined as $pcc = [i, bp, CR, cca, ape]$, where *i* is a URN, *bp* is the URN of the business process, *CR* is the set of authorization credentials to be obtained from the Certificate and Credential Authority, *cca* to get authorized to invoke *bp*. The *ape* is the URN of the Authorization Policy Evaluator that requires these credentials. This means each *bp* can have one or more of these (tuple) entries in the ARD.

The BusinessProcess-Attribute-DAS tuple (PATD tuple) is defined as *patd* = [*i*, *bp*, *AT*, *das*, *ape*], where *i* is a URN, *bp* is the URN of the business process, *AT* is the set of attributes to be obtained from a Dynamic Attribute Service, *das* to make an authorization decision. The *ape* is the URN of the Authorization Policy Evaluator that requires these attributes. This means each *bp* can have one or more of these (tuple) entries in the ARD.

C. BPAA Authorization Algorithms

The BPAA supports three authorization algorithms. The first, the *push-model* algorithm, supports the authorizations where a client's Client Proxy (CP), using the information in the BP-AuthorizationPolicy, collects and sends the required credentials (from the CCAs) and attributes (from the DASs) to a Business Process Security Manager (BPSM). The second, the *pull-model* algorithm, supports the authorizations where the Authorization Server (AZS) itself collects the required credentials from the CCAs, and the APEs collect the required attributes from the DASs. The AZS in this case uses the runtime objects information from the Authorization Runtime Database to be able to do so. The third, the *combination-model* supports both the push and pull models for collecting the required credentials and attributes. An organization must deploy one of these algorithms depending on the access control mechanisms used by the business process.

When the combination-model algorithm is deployed by an organization, the organization's Authorization Manager (AZM) may arbitrarily decide whether the credentials required from a CCA and dynamic attributes required from a DAS for each of the business process' APEs are fetched by a Client Proxy (push-model) or by the authorization components themselves (pull-model). The AZM may decide to give the entire responsibility of fetching the required credentials and attributes to the client proxy or to authorization components or share responsibility of fetching credentials and attributes amongst the client proxy and the authorization components. This information is reflected in a business process' BP-AuthorizationPolicy.

The BP-AuthorizationPolicy includes assertions that specify what credentials (and from which CCA) and attributes (and from which DAS) a client's Client Proxy has to collect before invoking a business process. These assertions also include the credentials and attributes required to invoke a *static* business process' partner Web services as well as its partner business processes. We extend the WS-BPEL statement schema to include the BP-AuthorizationPolicy. Note that the partner Web services and business processes-related authorization information is not included in the BP-AuthorizationPolicy of a dynamic business process. Such information is only necessary for a static business process. Finally, the authorization coordination information is also included in the BP-AuthorizationPolicy. This information is necessary only for the dynamic business processes. We have designed and implemented the authorization evaluation schemes for both static and dynamic business processes in push, pull and combined model scenarios. Due to lack of space in this paper, we refer the reader to [11] where they are described in full.

D. Dynamic Business Process Authorization

We leverage the WS-Coordination framework [12] to coordinate authorization of a client to a dynamic business process and its partner services. When a client invokes a dynamic business process, the Client Proxy component is responsible for the activation of a new instance of an Authorization Coordinator. It is aware that authorization coordination is required to get the client authorized to a dynamic business process, because the BP-AuthorizationPolicy has the information about the authorization coordinator, the coordination protocol used and its type (authorization coordination type), and finally its location. The Business Process Security Manager is another participant in the coordination protocol. During the course of execution of a dynamic business process, if the WS-BPEL Engine needs to invoke a partner service, it sends a message about the same, to the business process' BPSM. BPSM then informs the Authorization Coordinator that a partner service has been invoked and it needs authorization credentials from the client (Client Proxy). The Authorization Coordinator also informs the Client Proxy about the same. The Client Proxy fetches the required credentials and gets back to the Authorization Coordinator. The Authorization Coordinator then sends a message with the received credentials to the Business Process Security Manager. BPSM sends these credentials to the BPEL Engine. The BPEL engine uses these credentials and then continues execution of the partner service.

The Client Proxy interacts with the BPSM sending and receiving messages as normal, with the exception that it embeds the authorization coordination context (which carries the authorization information) in a SOAP header block in its messages to provide authorization credentials for those partner services (web services and/or business processes) that are invoked. Also the Client Proxy itself registers as a participant with the authorization coordinator. The BPSM understands the protocol messages associated with our authorization service. If it has not registered a participant previously, it does so once it receives a SOAP message from the Client Proxy containing an authorization context header using the details provided in the context (via the WS-Coordination registration service URI). This register operation occurs every time that the BPSM receives a particular context for the first time.

When the Client Proxy receives the final response from the BPSM after the execution of the business process, it sends a *Completion Message* to the Authorization Coordinator. The Authorization Coordinator then sends the Completion Message to the BPSM registered as a participant to the Authorization Coordinator. Any subsequent calls by the Client Proxy (on behalf of the client) to that business process with the same context will result in the service being unable to register a participant since the context details will no longer resolve to a live coordinator with which to register This is shown in Figure 4.

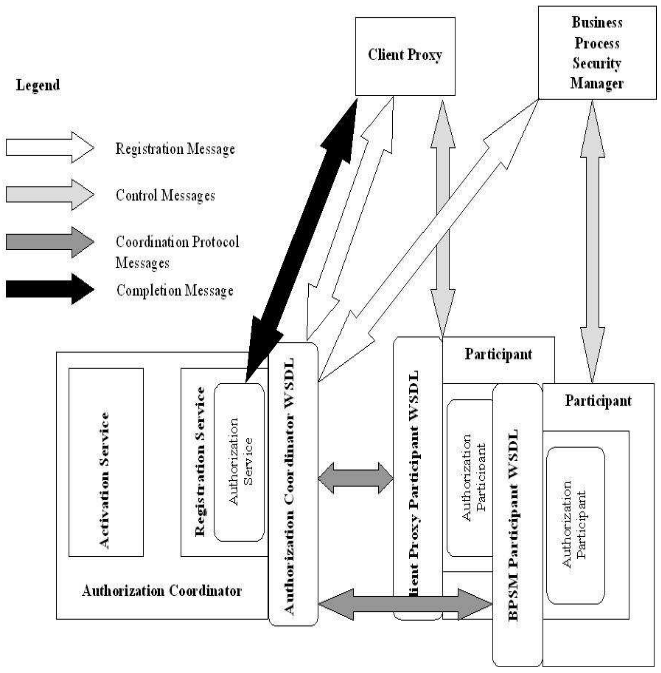
Legend
Registration Message
Control Messages
Coordination Protocol Messages
Completion Message
Client Proxy
Business Process Security Manager
Activation Service
Registration Service
Authorization Service
Authorization Coordinator WSDL
Authorization Coordinator
Client Proxy Participant WSDL
Participant
Authorization Participant
BPSM Participant WSDL
Participant
Authorization Participant

Figure 4: Authorization Coordination Framework

We have developed an implementation of this architecture using Microsoft BizTalk Server to create sample business processes in the BPEL4WS and demonstrated the features of the BPAA using those business processes. The architecture was used to demonstrate a healthcare application scenario using .NET framework. Performance evaluation showed that the BPAA architecture introduced an average performance delay of some 200ms when invoked with different processes.

IV. CONCLUDING REMARKS

In this paper, we have proposed and developed authorization architecture for business processes (BPAA) in SOA. Our authorization architecture is able to support both static and dynamic business processes. Also, a business process may have web services or even other business processes as partners. We took all such scenarios into consideration and have provided a comprehensive architecture for authorization for the business process layer of the SOA. Also we extended our authorization coordination framework to allow for both static and dynamic business processes to invoke partner services that are themselves dynamic business processes. The proposed BPAA supports multiple access control models including the MAC, DAC, and the RBAC models. Access control mechanisms can either use push or the pull model or even a combination of both, for collecting client credentials. Our architecture provides decentralized security administration. The partners involved in a business process workflow are allowed to autonomously control their authorization policies. The partners can be either from within an organization or from multiple organizations. In the case of static business processes, the information about the authorization credentials required to invoke the partner services is exposed in WS-BPEL using BP-AuthorizationPolicy at the design time itself. In the case of dynamic business processes, dynamic discovery and orchestration of business process' partners' authorization evaluation components are achieved. The BPAA coordinates the authorization where binding to real partner services happens at runtime depending on client requirements. Furthermore, The BPAA does not require the partner services to disclose their policies to the partner that is controlling the business process. The authorization of the client happens at the same place, where the partners originally intended it to be. Hence organizations can now leverage the services offered by the BPAA and do business by binding to the portal agents even if they do not trust them to perform client authorization. The Business Process Security Manager can be placed in a firewall zone, which enhances the security of business processes placed behind an organization's firewall. We have implemented the proposed architecture and its components using the .NET middleware platform and demonstrated its operation by developing a healthcare application scenario over this architecture.

REFERENCES

[1] T. Andrews et al., Business Process Execution Language for Web Services, http://www.ibm.com/developerworks/library/specification/ws-bpel/ (accessed Jan 2011).

[2] S. Agarwal, B. Sprick, and S. Wortmann, "Credential Based Access Control for Semantic Web Services," American Association for Artificial Intelligence, pp. 110-120, 2004

[3] R. Kraft, "Designing a Distributed Access Control Processor for Network Services on the Web", Proc of the ACM Workshop on XML Security, USA, pp. 36-52, 2002.

[4] H. Koshutanski and F. Massacci, "An Access Control System for Business Processes for Web Services," Informatica e Telecomunicazioni, University of Trento, Technical Report DIT-02-102, 2002

[5] M.C. Mont, A.Baldwin and J.Pato, "Secure Hardware-based Distributed Authorization underpinning a Web Service Framework", HPLabs Technical Report HPL-2003-144, 2004.

[6] S. Indrakanti, V.Varadharajan and R.Agarwal, "On the Design, Implementation and Application of an Authorization Architecture for Web Services", International Journal for Information and Communication Security, Vol.1, No.1/2, pp. 64-108, 2007.

[7] D.W. Chadwick and A.Otenko, "The PERMIS X.509 role based privilege management infrastructure", Future Gener. Comput. Syst. 19, pp. 277 - 289, 2003

[8] V. Varadharajan, C.Crall and J.Pato, "Authorization in Enterprise wide Distributed Systems: Design and Application", Proceedings of the 14th IEEE Computer Security Applications Conference, pp. 178-189, 1998.

[9] M.I. Yague and J.M.Troya, "A Semantic Approach for Access Control in Web Services". In Euroweb 2002 Conference. The Web and the GRID: from e-science to e-business, Oxford, UK, pp. 483-494, 2002.

[10] K. Beznosov et al., "A Resource Access Decision Service for ORBA-Based Distributed Systems," in Proceedings of the 15th Annual omputer Security Applications Conference: IEEE, pp.310-319, 1999.

[11] S. Indrakanti, "Engineering Authorization Services for Service Oriented Architectures, PhD Thesis, Macquarie University, 2007.

[12] L.F. Cabrera et al., "Web Services Coordination Framework", http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-tx/WS-Coordination.pdf (accessed Jan 2011).