Chapter 5

Synchronization of Authorization Flow with a Work Object Flow

This chapter presents the issue of synchronization of authorization flow with work object flow in a document production workflow environment. The chapter has shown how a work object flow is synchronized with the authorization flow using a central arbiter in Web service paradigms. The co-ordination of Web services is done using WS-BPEL which supports orchestration and XACML provides authorization for Web services. The synchronization is achieved by exploiting the obligation provisions in XACML.

5.1 Introduction

Synchronization of authorization flow with the workflow is a fundamental security requirement in a workflow environment. In case of automatic time-bound execution of tasks, issue of authorization flow is addressed by temporal constraints in the definition of tasks in the workflow. It is suitable for time-bound production workflows. In a general e-Business scenario, authorization flow is based on events not on temporal constraints. Authorizations for certain privileges are granted to a subject on some objects based on the occurrence of an event and revoked automatically on occurrence of the other conjugate event. The most common conjugate pair of events in an e-Business scenario is send and receive. As soon as the work object is sent to the next task, the current role and hence the task loses full or partial
privileges to the work object, whereas next role gains certain privileges on the work object to act upon by the task assigned to the role.

Let us take the case of an office. After receiving a request document, an office reacts to it, reactions are recorded as comments and finally a response document is sent to the requester. In a typical office, a worker, called the originator, creates a document and sends it to the another worker, who examines the case with respect to existing rules, regulations and precedents, adds his own comment along with the context and forwards it to the another worker and so on. As a result, we get a composite document, composed of an ordered list of comments contributed by the originator and the reviewers during the process. This workflow is termed as Document Production Workflow (DPW) [41]. The output of this workflow is the ordered list of comments, called Multi-Part Multi-Signature Document (MPMSD) [37]. In a MPMSD, the first part is the request document, the last part is the response document and the other parts in between are the reactions of the reviewers. A reviewer can read the previous parts but cannot modify the content of any of the previous parts, even cannot reorder or drop any of the previous parts [37]. The work object of such a workflow is a multi-part object. Each part is the output of a task. A role assigned to the task is granted authorization to execute the task only when the previous parts of the work object are in place. Previous user looses the write privileges as soon as the work object is sent to the next user. The present user gains the privileges to read previous parts and add a new part to the work object as soon as she receives the work object.

A comprehensive study to the MPMSD protocol with a central arbiter was done in [37]. The main aim of this chapter is to study the protocol in Web service domain, specially the issue of synchronization of authorization flow with the work object flow, both in architecture level and protocol level, in a Document Production Workflow (DPW) environment.

5.2 A Scenario

This is a Document Production Workflow (DPW) scenario common to many offices. An office worker, A submits an application, mA regarding his leave sanction for approval to the Head, B of the department, where he is working. B verifies the leave rules and gives his comment, mB and forwards it to the Registrar, C. C verifies all the leave status and gives his
comment, \( m_C \) on the application and forwards it to the Director, D. D approves the leave application with the addition of his approval note, \( m_D \). The multi-part document finally may go back to the originator A and the original multipart document is stored in a folder. The flow of the document is recorded in log-books. This is a case of the travel plan workflow. The DPW of the scenario is available in [41].

In a manual system, it is the same paper document that is passed around and the proof that it has come through the proper channel with the addition of series of comments followed by the signatures of the reviewers. As soon as an employee submits a travel plan request, the request document forms the first part of the work object. After sending the document to the next reviewer, the employee looses read, write privileges on the work object. The Head is granted to execute task verify leave status of the employee, add a part on the document and sends it to the next reviewer and so on.

### 5.3 Security Issues

The Security issues of DPW are like part integrity of a MPMSD, reuse of parts and authorization flow. Details of the issues like part integrity and reuse of parts are discussed in [37]. However, the issue of synchronized authorization flow is not clearly discussed. According to authorization flow, access right to a MPMSD moves from one reviewer to the next synchronized with the flow of the work object, i.e. MPMSD. Only the latest reviewer can read the document and can add comments to it. A reviewer can only read the previous parts. She is not allowed to modify the contents of any of the previous parts. Even the latest reviewer of a part should not be allowed to modify her own comment after signing and forwarding the document to the next reviewer. This chapter mainly focuses on this issue particularly in the parlance of Web Services. A protocol for MPMSD production with a central arbiter was proposed in [37], [38]. This protocol addresses the security issues raised in this section. The challenge in the Web Service domain is how to implement the protocol using the current industrial standards, like BPEL and XACML and ensure the synchronization of authorization flow with the MPMSD flow.
5.4 MPMSD Production Protocol

The responsibilities of generation, storage and flow control of MPMSD lies with a central, neutral, trusted and strong arbiter. First, an employee submits a travel plan request to the arbiter. The arbiter will verify the identity of the employee whether she is a valid user or not. As soon as the employee submits an application to the arbiter, she loses modify or delete privileges and gains read privilege depending on arbiter policy. The arbiter will present a multi-part document to a reviewer. The reviewer will submit her comment on the document to the arbiter along with the user-id of the next reviewer. The next reviewer can gain read privileges of previous comments as well as add her comment on the document but loses modify or delete privileges on the document. The arbiter will verify the comment and add it to the document as a part and then present the document to the next reviewer and so on. During the process it will also generate the non-repudiation evidences of submission of a comment and of delivery of a document. It is shown in the figure 5.4 (a), where $I_d$ is the unique identifier of the document given by N [37].

![Diagram of MPMSD](image)

Figure 5.4 (a): Generation of MPMSD
Summary: $A_i$ reviews the document $d_{i-1}$, submits her comment $m_i$ to $N$ and marks $A_{i+1}$ as the next reviewer. $N$ adds the comment $m_i$ to $d_{i-1}$ giving $d_i$ and then delivers the document $d_i$ to $A_{i+1}$, who in turn submits her comment $m_{i+1}$ to $N$ which has shown in figure 5.4 (b) [37].

![Figure 5.4 (b): MPMSD Protocol](image)

5.4.1 Protocol Steps

The protocol steps for MPMSD production are as follows:

1. $A_i \rightarrow N : \{ A_i, m_i, I_d, A_{i+1}, d_i \}$
2. $N \rightarrow A_i : \{ N, d_i \}$
3. $N \rightarrow A_{i+1} : \{ N, d_i, m_i \}$
4. $A_{i+1} \rightarrow N : \{ A_{i+1}, d_i \}$
5. $N \rightarrow A_{i+1} : \{ N, d_i \}$
6. $A_{i+1} \rightarrow N : \{ A_{i+1}, m_{i+1}, I_d, A_{i+2} \}$
Actions at Each Step

1. $A_i \rightarrow N$: The originator $A_i$ submits a document $d_i$ with her comment $m_i$ and marks it to the next reviewer $A_{i-1}$ by using an identity $I_d$ of the next reviewer and then sends it to the arbiter $N$.

2. $N \rightarrow A_i$: The arbiter $N$ then verifies the originator $A_i$ whether she is a valid user or not for accessing the document $d_i$.

3. $N \rightarrow A_{i+1}$: The arbiter $N$ then sends the document $d_i$ to the next reviewer $A_{i+1}$ with the comment $m_i$.

4. $A_{i+1} \rightarrow N$: The next reviewer $A_{i+1}$ sends a request to the arbiter $N$ for the permission of accessing the document $d_i$.

5. $N \rightarrow A_{i+1}$: The arbiter $N$ then verifies the identity $I_d$ of the next reviewer $A_{i+1}$ and then gives permission to access the document $d_i$ which is sent by the originator $A_i$.

6. $A_{i+1} \rightarrow N$: The next reviewer $A_{i+1}$ submits her comment $m_{i+1}$ on the document $d_i$ and marks it to the next reviewer $A_{i+2}$ with an identifier $I_d$ and so on.

5.5 WS-BPEL for DPW

In general, Web service, under Service Oriented Architecture (SOA) framework, is the state of the art technology for implementing workflow. The co-ordination of Web services is done in two ways: either by choreography or by orchestration. WS-BPEL is the standard which supports orchestration. The central arbitration mechanism discussed in section 5.4 is a form of orchestration. Therefore, an ideal technology to implement DPW with central arbitration mechanism is WS-BPEL. Here, WS-BPEL process works as arbiter $N$ and Web services implement different review processes accomplished by different reviewers as mentioned in the protocol. WS-BPEL process co-ordinates all the Web services and also controls the flow of the work object.

Business Process Execution Language (BPEL) is an XML-based language to specify business processes that orchestrate the operations of several Web services. WS-BPEL is
designed to describe business processes in a structured way. The business logic is expressed as a group of activities and performed by invoking Web services [48]. The <process> element is on the top level of BPEL specification. The attributes of <process> element specifies the process name and related namespaces being referred to. The <partnerLinks> element indicates the external Web services to be invoked from this process. The <variable> element defines the data variables involved in this process. The <sequence> element contains sequentially executed set of activities. Each step in a WS-BPEL process is called an activity. The <invoke> element that allows the business process to invoke a one way or both way (request-response) operation offered by a partner. The elements may contain one or more basic elements such as <receive> and <reply> element which define the message flow in a process. A synchronous BPEL process is one which we call and wait for the reply before proceeding further. On the other hand, an asynchronous BPEL process is one which we call and instead of waiting for the reply continue further proceeding. Both the processes first wait for the initial message using a <receive> element. A synchronous BPEL process will return a response after completing the process. Therefore, we use a <reply> element at the end of the process. But, an asynchronous BPEL process does not use the <reply> element. If such a process has to send a reply to the client, it uses the <invoke> element to invoke the callback operation on the client. In WS-BPEL process for DPW, an originator sends a request to the arbiter and waits for the response until and unless arbiter gives reply. After getting reply from arbiter, the originator will proceed further for the next process. The scenario that discussed in section 5.2 is synchronous only. However, in other some scenarios of DPW, it can be asynchronous as well. The detail of WS-BPEL specification is available in [51].

In the travel plan scenario the whole process can be divided into five activities:

- The employee applies for Travel Plan (travelRequest).
- Head verifies the leave rules on employee's travel plan request (travelAReview).
- Registrar verifies the leave status on originator's travel plan (travelBReview).
- Director approves the travel plan (travelApprove).
- The application response is sent back to the originator (travelResponse).
The process orchestrates the operations of four Web services:

- A Web service that provides the operation (travelRequest) to apply the travel plan.
- A Web service that provides the operation (travelAReview) to verify leave rules.
- A Web service that provides the operation (travelBReview) to verify the leave status.
- A Web service that provides the operation (travelApprove) to approve the travel plan decision.

The WS-BPEL implementation for DPW can be worked out as discussed in the following. The <partnerLinks> element indicates the participators or reviewers in the travel plan application process. The <variable> element defines the variable name as 'travelRequest' for travel plan application. In <sequence> element, all the steps involved in travel plan request will be invoked in an ordered sequence. An employee submits a travel plan request to the arbiter. The <receive> element receives a travel plan request from the employee. The <assign> element will manipulate the data variables and the <copy> element will copy data between <from> and <to> element by assigning the variable ‘travelRequest’ to any other variable. The reviewerA review the travel plan request invoked by the <invoke> element on the employee’s document. The <assign> and <copy> element will concatenate the variables ‘travelRequest’ and ‘travelAReview’ using ‘concat’ function. The reviewerB review the travel plan request which is invoked by the <invoke> element. The <assign> and <copy> element will concatenate the variables ‘travelAReview’ and ‘travelBReview’ using the ‘concat’ function. The approver approves the travel plan request. The approver makes a decision for the travel plan request which is further invoked by the <invoke> element. The <assign> and <copy> element will concatenate variables ‘travelBReview’ and ‘travelApprove’ using ‘concat’ function. Finally, the travel plan response is sent back to the employee by the <reply> element.
5.6 XACML for Authorization Flow

The eXtensible Access Control Markup Language (XACML) is an XML based language which is required to make authorization decisions. The decision may be permit, deny, indeterminate or error. The XACML architecture [47] specifies the implementation of a Policy Enforcement Point (PEP), which is an entity that performs access control by enforcing authorization decision. A Policy Access Point (PAP) is an entity that creates policies or policy sets. A Policy Decision Point (PDP) is an entity that evaluates applicable policy or policySets to renders an authorization decision. A Policy Information Point (PIP) is an entity which is having information about the attributes of subject, resource, action and environment. The Context Handler receives the access request from PEP and converts the access request context to the XACML context. The data flow model of XACML architecture is as follows –

At first, the PAP creates a policy. At request time, an access request arrives at the PEP and is send to the Context Handler. The Context Handler determines resources to be accessed, collects all required attributes of subject, resource, action and environment from PIP and forwards them to the PDP. PDP then acquires the policy from the PAP, evaluates the relevant policy and returns the access decision to the PEP through Context Handler which proceeds to enforce the authorization decision. The PDP sends the authorization decision to the PEP through Context Handler with some obligations. The PEP fulfils the obligations. If access is permitted, then the PEP permits access on the resource; otherwise, it denies access [23], [50].

The authorization flow using XACML in DPW play an important role while grant and revoke privilege on a work object. Considering the example of travel plan application, during authorization flow a reviewer is granted read and adds comment while she is revoked write, modify or delete privileges. An employee A_i submits a travel plan request to the arbiter N. The arbiter N will first verify the identity of A_i whether she is a valid user or not. As soon as the travel plan request is submitted to the arbiter, the employee is revoked modify or delete privileges. She may be granted read privileges depending on the arbiter policy. The next reviewers such as reviewerA and reviewerB are granted read previous comments and add their own comments on the travel plan request where they are revoked modify or delete privileges. Similar is the case of approver where she is granted to verify all the previous comments and give his own comment for approval but she is also revoked modify or delete privileges.
5.6.1 XACML Protocol

The protocol steps for XACML architecture are as follows:

1. PAP \rightarrow PDP : \{PAP, ps_i\}
2. A_i \rightarrow PEP : \{A_i, a_{req}, r_1\}
3. PEP \rightarrow CH : \{PEP, a_{req}, r_1\}
4. CH \rightarrow PDP : \{CH, a_{req}, xacml_{req}\}
5. PDP \rightarrow CH : \{PDP, s_{attr}, r_{attr}, a_{attr}\}
6. CH \rightarrow PIP : \{CH, s_{attr}, r_{attr}, a_{attr}\}
7. PIP \rightarrow CH : \{PIP, s_{attr}, r_{attr}, a_{attr}\}
8. CH \rightarrow PDP : \{CH, s_{attr}, r_{attr}, a_{attr}\}
9. PDP \rightarrow CH : \{PDP, r_1, a_{dec}\}
10. CH \rightarrow PEP : \{CH, r_1, a_{res}\}

**Actions at Each Step**

1. PAP \rightarrow PDP: The PAP writes policySets ps_i and makes them available to the PDP.
2. A_i \rightarrow PEP: An originator or reviewer A_i sends an access request a_{req} to add to the PEP for access an object r_1.
3. PEP \rightarrow CH: The PEP then sends the access request a_{req} to the Context Handler for an authorization decision.
4. CH \rightarrow PDP: The Context Handler converts the access request a_{req} to an XACML request xacml_{req} and sends it to the PDP.
5. PDP \rightarrow CH: The PDP then sends a request to the Context Handler for the information about the attributes of subject, resource and action (s_{attr}, r_{attr}, a_{attr}).
6. CH → PIP: The Context Handler then sends a request to the PIP for information about the attributes of subject, resource and action (s_{attr}, r_{attr}, a_{attr}).

7. PIP → CH: The PIP obtains information about the attributes of subject, resource and action (s_{attr}, r_{attr}, a_{attr}) and then returns them to the Context Handler.

8. CH → PDP: The Context Handler then sends the information about the attributes of subject, resource and action (s_{attr}, r_{attr}, a_{attr}) to the PDP which will be matched with the policySet ps, available in PDP.

9. PDP → CH: The PDP then returns an authorization decision a_{dec} on the object r_{1} to the Context Handler.

10. CH → PEP: The Context Handler finally returns the access response a_{res} on an object r_{1} to the PEP. The PEP fulfils the obligations. If access is permitted, then the PEP permits access on the object r_{1}; otherwise, it denies access.

5.7 Synchronization

In the earlier sections of this chapter, the work object flow using BPEL where the flow of an object is controlling by the WS-BPEL process is discussed. The authorization flow using XACML where an authorization decision is given to an access requester is also discussed. The reviewers can access the work object by applying the granted and revoked privileges. Now, in this section, the solution of synchronization of authorization flow with a work object flow has shown in two levels – the architecture level and the protocol level. In the conceptual architecture level, a 3-tier architecture is proposed. In the protocol level, the merging of the protocol for work object flow and the protocol for authorization flow are discussed.

5.7.1 Architecture

A three tier architecture consists of three layers namely - presentation layer, logic layer and data layer. The presentation layer consists of view which contains GUIs (Graphical User Interfaces), the logic layer consists of processes to manipulate data and data layer stores data. The logic layer is again split into two – client logic and arbiter logic because the data
manipulation will be done by the client and the arbiter co-operatively. The client and the arbiter will communicate over a network using some protocols. The components of the architecture are described below –

The View

This part of the architecture gives basically the GUIs (Graphical User Interfaces) [36] of the system. View comprises of various interfaces through which a reviewer can interacts with the client logic module to review a document and adds own comment and forwards it to the next reviewer. Through this tier a user interacts with the workflow agent of the client logic. The interaction may be different for different roles.

The Client Logic

This module is populated by client side objects with attributes and methods. The objects in this module are termed as agent. The main agent found in this module is –

MPMSD client: This agent is responsible for client activities contain an object which submits an access request to the MPMSD arbiter for authorization decision. The responsibilities of this agent are – request the arbiter to get authorization decision, get the response from the arbiter as an authorization decision, and display the document in the GUIs module.

The Arbiter Logic

The arbiter is the central hub of the architecture. It certifies and authenticates subjects and objects, manages storage and retrieval of objects, manages authorizations of subjects on objects, time-stamping etc. The services of the arbiter are provided by the following components –

MPMSD arbiter: This component includes – forwarding a MPMSD to the next reviewer (client), addition of the part submitted by the client to the MPMSD under production, sending evidence of submission (NRS) of parts to the client, recording document flow in the logbooks, time-stamping the evidences of occurrences of events in the communication (sending, receiving, authoring etc.), authorization flow management during the communication with other components. In our scenario, since MPMSD arbiter is acting as a
TTP (Trusted Third Party), so without any loss of generality, in addition, it can also function as PEP and obligation handler. This is how synchronization of authorization flow with work object flow can be achieved.

*Storage Manager:* This component stores and retrieves the document in and from the data storage.

The components like CH, PDP, PAP and PIP are as in the standard XACML architecture which is already discussed in section 5.6.

---

**Figure 5.7: The Three Tier Architecture**
The Data Storage

This layer is basically a repository of office objects [35]. MPMSD storage stores multi-part documents which is managed by storage manager. The Target storage stores the attributes of subjects, resources and actions along with their values as per the XACML architecture which is managed by PIP. Policy storage stores authorization policies managed by PAP.

5.7.2 Protocol

The synchronized protocol steps of both XACML and DPW are as follows:

1. $A_i \rightarrow N : \{A_i, add, d_i, m_i\}$
2. $N \rightarrow CH : \{N, A_i, add, d_i, m_i\}$
3. $CH \rightarrow PDP : \{CH, A_i, add, d_i, m_i\}$
4. $PDP \rightarrow CH : \{PDP, A_i, d_i, add\}$
5. $CH \rightarrow PIP : \{CH, A_i, d_i, add\}$
6. $PIP \rightarrow CH : \{PIP, A_i, d_i, add\}$
7. $CH \rightarrow PDP : \{CH, A_i, d_i, add\}$
8. $PDP \rightarrow PAP : \{PDP, A_i, d_i, add\}$
9. $PAP \rightarrow PDP : \{PAP, psi\}$
10. $PDP \rightarrow CH : \{PDP, permit/ deny, obl\}$
11. $CH \rightarrow N : \{CH, permit/ deny, obl\}$
12. $N \rightarrow S_m : \{N, add, d_i, m_i\}$
13. $S_m \rightarrow N : \{S_m, OK\}$
14. $N \rightarrow PDP : \{N, (A_i, revoke, add), (A_j, grant, add)\}$
Actions at Each Step

1. $A_i \rightarrow N$ : The MPMSD client $A_i$ submits to the MPMSD arbiter $N$ the comment $m_i$ to be added to document $d_i$.

2. $N \rightarrow CH$ : The MPMSD arbiter $N$ then requests the Context Handler for an authorization decision.

3. $CH \rightarrow PDP$ : The Context Handler sends the XACML request $add$ to the PDP for authorization decision.

4. $PDP \rightarrow CH$ : PDP sends a request to the Context Handler for collecting the information about the attributes of subject, resource and action ($A_i$, $d_i$, add).

5. $CH \rightarrow PIP$ : The Context Handler then sends the request to the PIP for information about the attributes of subject, resource and action ($A_i$, $d_i$, add).

6. $PIP \rightarrow CH$ : The PIP gives all the information about the attributes of subject, resource and action ($A_i$, $d_i$, add) to the Context Handler.

7. $CH \rightarrow PDP$ : The Context Handler then gives the information about the attributes of subject, resource and action ($A_i$, $d_i$, add) to the PDP.

8. $PDP \rightarrow PAP$ : The PDP requests for policySets $ps_i$ to the PAP.

9. $PAP \rightarrow PDP$ : The PAP then writes policySets $ps_i$ to the PDP which will be matched with the attributes of subject, resource and action ($A_i$, $d_i$, add) for an authorization decision.

10. $PDP \rightarrow CH$ : After matching the attributes with the policySets $ps_i$, the PDP sends an authorization decision either permit or deny with some obligations to the Context Handler.

11. $CH \rightarrow N$ : The Context Handler then sends the authorization decision either permit or deny with some obligations to the arbiter $N$.

12. $N \rightarrow S_m$ : The MPMSD arbiter $N$ then submits the comment $m_i$ to be added to the document $d_i$ to the storage manager $S_m$. 
13. $S_m \rightarrow N$ : The storage manager $S_m$ then gives an OK report to the arbiter $N$.

14. $N \rightarrow PDP$ : Finally, the arbiter $N$ requests the PDP to revoke from $A_i$ the same privilege to $A_j$, the next reviewer already predefined in the arbiter. PDP revokes and grants accordingly.

5.8 Chapter Summary/Discussion

This chapter discusses the issue of synchronization of authorization flow with the work object flow in a workflow problem in general and DPW in particular. The synchronization is shown in this chapter in architecture level, protocol level exploiting obligation mechanism available in XACML standard. It is evident from the above discussion is that a central arbiter is needed to give the solution of synchronization of authorization flow with a work object flow. The discussion is limited to synchronization of orchestrated web services with XACML only. However, synchronization of choreographed Web services with XACML is another interesting area to be explored. This remains to be the future direction of research.