Chapter 6

Extra-Tree: A Model to Organize Execution Traces of Web Services

In this chapter, a non-linear model called Extra-Tree is proposed to organize execution traces of orchestrated Web services. This proposed model provides us a logging system which records the history of all activities from the initiation to the completion of a Web service. The main focus of this chapter is to organize execution traces of Web services in a distributed computing paradigm in the form of a tree. One of the special characteristics of this model is that the execution traces of Web services can be retrieved from the coarse-grained level to the fine-grained level of the tree as per the requirement.

6.1 Introduction

XML security standards, such as WS-security [30], SAML [67], XML signature [22] and XML encryption [68] are powerful tools for people to build security architectures in distributed systems to deliver message level authentication, integrity and encryption services. These standards offer new and robust ways to protect Web services, but they don’t provide services for audit logging [56]. An interesting challenge is how to handle logging in distributed environments such as those of Web services. Web services architecture should have its own logging mechanism, so that each component involved in that particular architecture has a uniform way of creating a log for later access from a single point. The best way to perform logging in Web services architecture might be a fully distributed logging
mechanism. Both successful and failed attempt to get access to data or services must be logged. The net result of the Web services logging problem is both a challenge and an opportunity [56].

Audit logging is a vast area of research. The spectrum of types of data to be logged is wide. Depending on the perspectives or the requirements, like security, system management, event analysis etc., different types of data are to be logged. The scope of this chapter is limited to logging execution traces of orchestrated Web services. This chapter proposes a model to organize execution traces of orchestrated Web services in a tree structure, named Execution Trace Tree, Extra-Tree in short. In the proposed model each Web service involved in a composition has a uniform way of creating a log of execution traces for later access from a single point. The execution traces are logged in a distributed way. A special characteristic of the hierarchical model is that execution traces can be accessed in varying granularities from coarse-grain to fine-grain.

6.2 Audit Trail

An audit trail is a record showing who has accessed a computer system and what operations she has performed during a given period of time [55]. Audit trails are useful both for maintaining security and for recovering lost transactions [52], [53], [54]. Audit trail is a sequential record of computer system activities saved to a file on the system [59]. Secure logging [57] means the time of access, when it has accessed, who has accessed, what has accessed and their details are traced or logged. In non-repudiation logging [58], once a user is accessing a work object, next time it cannot be denied which has already traced. A secure audit log is also required for non-repudiation [58].

Audit trails are basically the protocol about what happened at a specific service. Audit trails are providing the following capabilities –

1) They guarantee non-repudiation, i.e. if dealing with a customer order, the manufacturer can prove that a certain order was placed by a certain principal.
2) They also give the requester a tool by which he can review (double check) his orders.
3) Audit trails should provide enough qualitative information to re-build the business transaction in case of problems like attacks, disk crashes etc.
6.3 Execution Traces

Traces can contain different information and can contain different types of information depending on what is being traced and the purpose of the trace. An execution trace can be used to describe the interacting modules involved in a particular scenario or may be detailed to capture the performed statements in each module’s procedure.

The execution traces form a subset of audit trails. In execution trace only the current executable data are recorded from the time of initiation to the time of completion. The information about the system activities is typically represented using execution traces. The system activities such as login and logout, insertion, deletion, modification, date and time of action are normally traced. Execution traces represent the sequence of execution in a running software system at different levels of abstraction [65], [66]. Traces may exist in different structures according to the degree of abstraction requested by the program analyzer. Routine-level traces depict the sequence of routine calls for a program and often represented as a tree structure. The execution traces need to be captured in different levels: coarse-grain level to the fine-grain level and resources accessed during operations. This multi-level capturing of traces is useful for security issues like authorization, auditing, non-repudiation etc.

![Figure 6.3: Execution Traces](image-url)
6.4 BPEL Process

BPEL (Business Process Execution Language) is a language for Web services to model Web services based business processes. The core concept is the representation of peer-to-peer interactions between a process and its partners using Web services and an XML-based grammar. It is built on top of WSDL. BPEL is a language based on XML that allows control of the process flow of a set of collaborating Web services. It defines interactions that exist within and between organization processes. The language uses either a graph based or algebraic representation and offers the ability to manage both abstract and executable processes. BPEL offers an interesting feature that allows independent representation of the interactions between the partners. The interaction protocols are called abstract processes and they are specified in business protocols.

BPEL basic activities are handled by three types of messages. These are –

1) `<invoke>`: to make an operation on a partner.

2) `<receive>`: to receive an invocation from a partner.

3) `<reply>`: to send a reply message in partner invocation.

The `<assign>` activity is used to copy data from one variable to another.

BPEL is today the state of the art technology solution for Web Service orchestration. BPEL specifies a composite Web Service composed of other Web services or BPEL processes or the both. It's basically an integration artifact for integrating Web services. BPEL engine accomplishes the task as per the BPPEL specification of the business process. An important issue is to model the execution traces of BPEL process which comprises of the local activities within the BPEL process as well as remote Web services linked in partner links. The execution traces need to be captured in different levels: process, activity, operations and resources accessed during operations. This multi-level capturing of traces is useful for authorization auditing, non-repudiation, dependability analysis etc. In this chapter, it has proposed a tree model for execution traces of BPEL processes which facilitates multi-level trace analysis and which can be implemented easily both in centralised as well in distributed version using XML. More details about WS-BPEL are available in [29].
6.5 Related Works

In the paper [60], authors have given more importance of how the old information is either deleted or stored in the audit trail in a systematic manner like the current data. In addition to the actual recording of all events that takes place in the database, an audit trail must also provide query support for auditing. In this paper [60], authors used a database activity model in which relational database technique is used. The benefit of using this model is that it not only stores the current data but also gives more importance on past information which is either deleted or stored in the database in a systematic manner. Another benefit of using this model is to providing mechanism for recording and querying accesses to the database. Pau et al. proposed [53] a Data Warehouse model for capturing audit trail data and analysis. The benefits of using these models is that the managers can evaluate quality of a business processes by monitoring the performance of the operations. From this, managers and decision makers can improve existing workflow models and can optimize the performance of the business processes. In the paper [61], authors proposed a framework for secure logging in a public communication network system which are more resilient to identify security threats and also responsible to verify the integrity of the log files. In the paper [62], authors proposed several issues which have resolved for a secure logging system. These issues are related to secure logging of decentralized cross-organizational workflows and underlying decentralized data storage. Fedaghi and Mahdi [52] proposed a conceptual model that produces an implementation independent logging scheme to monitor events. In this paper [52], authors proposed a model known as Flow-Based Logging Model which provides a conceptual foundation for the classification of information related events. Helmen and Liepins [63] presented a statistical model for the detection of computer misuse using stochastic process. The benefit of using this model is to give perfect information against computer misuse detection. In the paper [64], authors presented a new approach to providing audit logs for transaction processing systems that can effectively and efficiently detect tempering.

From the above study, it is found that the existing secure logging systems mainly protect the log files from external attacks and also it can give the information about suspicious or malicious activities. But the secure logging system of Web services in the distributed computing system is not much available in the literature. So, a tree model is proposed to organize and retrieve execution traces of Web services from coarse-grain level to fine-grain level.
6.6 Extra-Tree Model

This section discusses about the proposed Extra-Tree model to organize execution traces. The scope of this model is limited to orchestrated Web services only. Elementary Web services may be implemented in any platform but can be called through its published WSDL interface. These elementary Web services may participate in many composite Web services. Thus a composite Web service is composed of one or more Elementary Web services. The composite Web services may be implemented in BPEL. Each composite Web service has also a WSDL interface through which it can be called and thereby can participate in many composite Web services. Thus a composite Web service may be composed of other composite Web services or elementary Web services. Composition and participation creates a many-to-many relationship. Web service is a distributed computing paradigm in which capturing the many-to-many relationship in a central table is against the fundamental characteristic of the paradigm. In this model, it is assumed that each Web service has two lists: a Participation List (PL) and a Composition List (CL). The PL of a service contains the unique identifiers, like URIs, of the Web services where the service in question has participated in. For a top level service the PL may be null. Similarly CL of a service contains the identifiers of the services participated in the composition of the service in question. The CL of an elementary service is null. In addition to these two lists, each service has an Execution Trace Table (ETT), which stores the execution traces of the concerned Web service. Execution trace contains the attribute values of Transaction Number, who called the Web service (another service, role or user), when called and when the call ended etc. For simplicity of the discussion, we exclude other attributes like resources used during the instance etc. But such additional audit trails can be easily added without any loss of generality.

Now, consider any top level service and track down according to the composition lists until we find elementary services, the process yields a tree. The top level service is the root, elementary services constitute the leaves and intermediate nodes are composite services.
An interesting feature of Extra-Tree model is that the execution traces of Web services are organised in the ETTs of different levels of the tree model with varying granularities. Parent node contains the coarser-grain traces than the children, or in other words, children contain finer-grain traces than the parent node. This multi-level organization enables us to retrieve the execution traces in required details only. Unless finer details are required, ETTs stored in nodes of lower depths will satisfy the query requirement. Only when complete execution history is required, the tree is traversed in full. If the knowledge is considered at best, this is the first non-linear hierarchical model to organize execution traces in Web services domain.

In Figure 6.6 (a), root Web service is WS00, the execution traces in this Web services can be organized as -

\[
CL_{00} = (WS_{10}, WS_{11}, WS_{12}) \quad \text{and} \quad PL_{00} = NULL, \quad ETT_{00} = \{(T, U, t_{i}^{00}, t_{c}^{00}), \ldots, \ldots\}, \quad (\ldots)\}
\]

Where, \(CL_{00}, PL_{00}\) and \(ETT_{00}\) are the composition list, participation list and execution trace table of the root Web service \(WS_{00}\) and \(T\) is the transaction number, \(U\) is who called and \(t_{i}^{00}\), \(t_{c}^{00}\) are the time of initiation and time of completion of the transaction.
The composite Web services in Figure 6.6 (a) are WS10, WS11 and WS12. The execution traces in this Web services can be organized as –

\[
\begin{align*}
CL_{10} &= (WS_{20}, WS_{21}, WS_{22}), \quad PL_{10} = (WS_{00}, WS_{01}, WS_{02} \ldots ) \quad \text{and} \quad ETT_{10} = \{(T, WS_{00}, t_i^{10}, t_c^{10} \ldots ), (\ldots )\}, \\
CL_{11} &= NULL, \quad PL_{11} = (WS_{00}, WS_{01}, WS_{02} \ldots ) \quad \text{and} \quad ETT_{11} = \{(T, WS_{00}, t_i^{11}, t_c^{11} \ldots ), (\ldots )\}, \\
CL_{12} &= (WS_{20}, WS_{21}, WS_{22}), \quad PL_{12} = (WS_{00}, WS_{01}, WS_{02} \ldots ) \quad \text{and} \quad ETT_{12} = \{(T, WS_{00}, t_i^{12}, t_c^{12} \ldots ), (\ldots )\},
\end{align*}
\]

Where, \( CL_{10} \ldots \ldots CL_{12}, \ PL_{10} \ldots \ldots PL_{12} \) and \( ETT_{10} \ldots \ldots ETT_{12} \) are the composition list, participation list and the execution trace table of the composite Web services WS10, WS11 and WS12. Here T is the transaction no., WS00 is who called and \( t_i^{10}, t_c^{10} \ldots \ldots t_i^{12}, t_c^{12} \) are the time of initiation and time of completion of the transaction.

The elementary Web services in Figure 6.6 (a) are WS20, WS21, WS22, WS23 and WS24. The execution traces in this Web services can be organized as –

\[
\begin{align*}
CL_{20} &= NULL, \quad PL_{20} = (WS_{10}) \quad \text{and} \quad ETT_{20} = \{(T, WS_{10}, t_i^{20}, t_c^{20} \ldots ), (\ldots )\}, \\
CL_{21} &= NULL, \quad PL_{21} = (WS_{10}) \quad \text{and} \quad ETT_{21} = \{(T, WS_{10}, t_i^{21}, t_c^{21} \ldots ), (\ldots )\}, \\
CL_{22} &= NULL, \quad PL_{22} = (WS_{10}) \quad \text{and} \quad ETT_{22} = \{(T, WS_{10}, t_i^{22}, t_c^{22} \ldots ), (\ldots )\}, \\
CL_{23} &= NULL, \quad PL_{23} = (WS_{12}) \quad \text{and} \quad ETT_{23} = \{(T, WS_{12}, t_i^{23}, t_c^{23} \ldots ), (\ldots )\}, \\
CL_{24} &= NULL, \quad PL_{24} = (WS_{12}) \quad \text{and} \quad ETT_{24} = \{(T, WS_{12}, t_i^{24}, t_c^{24} \ldots ), (\ldots )\},
\end{align*}
\]

Where, \( CL_{20} \ldots \ldots CL_{24}, \ PL_{20} \ldots \ldots PL_{24} \) and \( ETT_{20} \ldots \ldots ETT_{24} \) are the composition list, participation list and the execution trace table of the elementary Web services WS20, WS21, WS22, WS23 and WS24. Here, T is the transaction no. WS10 and WS12 are who called and \( t_i^{20}, t_c^{20} \ldots \ldots t_i^{24}, t_c^{24} \) are the time of initiation and time of completion of the transaction.

For a given transaction T, the PLs, CLs and ETTs are tabulated as follows –
Table 6.6 (a): Execution Traces of Web service WS_{00}

<table>
<thead>
<tr>
<th>Who Called</th>
<th>Transaction Number</th>
<th>When Called</th>
<th>When Ended</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>T</td>
<td>$t_i^{00}$</td>
<td>$t_e^{00}$</td>
</tr>
</tbody>
</table>

Table 6.6 (b): Execution Traces of Web service WS_{10}

<table>
<thead>
<tr>
<th>Who Called</th>
<th>Transaction Number</th>
<th>When Called</th>
<th>When Ended</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS_{00}</td>
<td>T</td>
<td>$t_i^{10}$</td>
<td>$t_e^{10}$</td>
</tr>
</tbody>
</table>
**WS\(_{11}\)**

<table>
<thead>
<tr>
<th>PL(_{11})</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WS(_{00})</strong></td>
</tr>
<tr>
<td><strong>CL(_{11})</strong></td>
</tr>
<tr>
<td><strong>NULL</strong></td>
</tr>
<tr>
<td><strong>ETT(_{11})</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Who Called</th>
<th>Transaction Number</th>
<th>When Called</th>
<th>When Ended</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WS(_{00})</strong></td>
<td><strong>T</strong></td>
<td>(t_i^{11})</td>
<td>(t_e^{11})</td>
</tr>
</tbody>
</table>

*Table 6.6 (c): Execution Traces of Web service WS\(_{11}\)*

**WS\(_{12}\)**

<table>
<thead>
<tr>
<th>PL(_{12})</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WS(_{00})</strong></td>
</tr>
<tr>
<td><strong>CL(_{12})</strong></td>
</tr>
<tr>
<td><strong>WS(_{23})</strong></td>
</tr>
<tr>
<td><strong>ETT(_{12})</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Who Called</th>
<th>Transaction Number</th>
<th>When Called</th>
<th>When Ended</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WS(_{00})</strong></td>
<td><strong>T</strong></td>
<td>(t_i^{12})</td>
<td>(t_e^{12})</td>
</tr>
</tbody>
</table>

*Table 6.6 (d): Execution Traces of Web service WS\(_{12}\)*
**Table 6.6 (e): Execution Traces of Web service WS_{20}**

<table>
<thead>
<tr>
<th>Who Called</th>
<th>Transaction Number</th>
<th>When Called</th>
<th>When Ended</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS_{10}</td>
<td>T</td>
<td>t_{20}</td>
<td>t_{e_{20}}</td>
</tr>
</tbody>
</table>

**Table 6.6 (f): Execution Traces of Web service WS_{21}**

<table>
<thead>
<tr>
<th>Who Called</th>
<th>Transaction Number</th>
<th>When Called</th>
<th>When Ended</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS_{10}</td>
<td>T</td>
<td>t_{21}</td>
<td>t_{e_{21}}</td>
</tr>
</tbody>
</table>
### Table 6.6 (g): Execution Traces of Web service WS₂₂

<table>
<thead>
<tr>
<th>Who Called</th>
<th>Transaction Number</th>
<th>When Called</th>
<th>When Ended</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS₁₀</td>
<td>T</td>
<td>t₁²₂</td>
<td>tₑ²₂</td>
</tr>
</tbody>
</table>

### Table 6.6 (h): Execution Traces of Web service WS₂₃

<table>
<thead>
<tr>
<th>Who Called</th>
<th>Transaction Number</th>
<th>When Called</th>
<th>When Ended</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS₁₂</td>
<td>T</td>
<td>t₁²₃</td>
<td>tₑ²₃</td>
</tr>
</tbody>
</table>
Table 6.6 (i): Execution Traces of Web service WS24

For a given transaction, symbolically the parenthesis structure can be written as –

\[
\text{When called} \quad ( \quad \text{When ended} \quad ) \\
\text{Who called} \quad ( \quad \text{Who called} \quad )
\]

Figure 6.6 (b): The Parenthesis Structure

Therefore, the full execution of a transaction in the form of Well-formed expression can be written as –

\[
\begin{align*}
\left( \begin{array}{c}
\ell_i^{10} \\
\ell_i^{11} \\
\ell_i^{12} \\
\ell_i^{20} \\
\ell_i^{21} \\
\ell_i^{22} \\
\ell_i^{23} \\
\ell_i^{24} \\
\end{array} \right) & = \left( \begin{array}{c}
\ell_c^{10} \\
\ell_c^{11} \\
\ell_c^{12} \\
\ell_c^{20} \\
\ell_c^{21} \\
\ell_c^{22} \\
\ell_c^{23} \\
\ell_c^{24} \\
\end{array} \right) \\
\big( \begin{array}{c}
ws_{10} \\
ws_{20} \\
ws_{21} \\
ws_{22} \\
ws_{23} \\
ws_{24} \\
ws_{11} \\
ws_{12} \\
\end{array} \big) & = \big( \begin{array}{c}
w_{s10} \\
w_{s20} \\
w_{s21} \\
w_{s22} \\
w_{s23} \\
w_{s24} \\
w_{s11} \\
w_{s12} \\
\end{array} \big)
\end{align*}
\]

Figure 6.6 (c): The Well-formed Expression
In the above expression, WS$_{00}$, WS$_{10}$ ..........WS$_{24}$ are the root web services, composite Web services and elementary Web services respectively and $t_i^{00}$, $t_e^{00}$ ........ are the required time interval in each of the level of Web services to organise and retrieve them from the coarse-grained level to the fine-grained level of the tree model. The motivation behind this expression is that it can be easily implemented in nested XML elements with attributes. Now, to generate this expression we have to consider Depth-First-Search technique.

Execution traces of Web services are retrieved from Extra-Tree by RetrieveTrace algorithm. It is essentially a modified Depth-First-Traversal (DFT) algorithm. For coarse-grain traces there is no need to traverse the full tree. Therefore, the depth to be traversed is to be given as input to the algorithm. Depth 0 means root only, any number greater than 0 is the absolute maximum depth. Actual depth of the tree is less than the input maximum depth implies the full tree traversal. The RetrieveTrace algorithm is given below.

**Algorithm RetrieveTrace(W, T, L)** //algorithm to retrieve execution traces

1. T: Transaction No, L: Depth //declaring global variables
2. l := 0; //initializing l
3. DFT_VISIT(W) // to start visit the root Web service W

**Algorithm: DFT_VISIT(W: WS)** // algorithm for Depth-First-Traversal

1. {S: WS;
2. mark[W] := visited // root Web service W is already visited
3. Trace := select (ETT(W), T); // who called, transaction no. etc. will be selected from ETT
4. Print W/C(t$_i$); // print the traces of service who called, time of initiation
5. Push(stack, )t$_e$); // push traces into the stack
6. l := l+1 // incrementing l
7. While (l <= L) do
8. \{ For each S in PL[W] from left to right order
9. If mark[S] = unvisited then
10. DFT_VISIT(S);
11. \}
12. $T_e := \text{pop((stack, )e)}$; // pop traces from the stack and stores in a temporary variable

13. Print $T_e$; // Finally, the Well-formed parenthesis is printed

14. }

Table 6.6 (j): Algorithm RetrieveTrace

6.7 Chapter Summary/Discussion

This chapter discusses an Extra-Tree Model which organizes and retrieves execution traces of Web services in different levels. The retrieval of execution traces of Web services is shown by an algorithm using Depth-First-Search technique. The advantage of this model is to retrieve the execution traces from the coarse-grained level to the fine-grained level in different time intervals. Normally, all execution traces are linear but in this model traces are non-linear.

From this chapter, it is assumed that the ETTs, PLs and CLs are securely stored in the concerned Web services and resilient to attacks. But from the security point of view it is a costly assumption. If audit data stored in a distributed way are compromised then the whole objective of audit analysis will be meaningless. Therefore, the organization of execution traces of Web services in a central point rather than in a distributed way will probably give the more secured auditing system and make the proposed model robust which will be more challenging as a future research work.