An XML Based Approach to Enforcing History-Based Separation of Duty Policies in Heterogeneous Workflow Environments

Carl Papenfus and Reinhardt Botha

Faculty of Computer Studies, Port Elizabeth Technikon, Port Elizabeth, South Africa
rbotha@computer.org

Abstract

In the computing world a new technology occasionally comes along, promising to make dramatic changes to the way computing tasks are performed. The Extensible Markup Language (XML) has been heralded as one such technology. XML promises to provide a universal metadata mechanism for defining, understanding and interchanging information between possibly heterogeneous systems.

This paper exploits this powerful promise of XML by examining how it can be used to enforce history-based separation of duty policies across heterogeneous workflow environments. A very brief overview of separation of duty policies is provided, whereafter the need for history-based separation of duty is motivated through an extensive case study. A solution based on XML baggage is proposed and it is shown how the solution would operate in the context of the case study.

Keywords: Workflow, Separation of duty, Access Control, XML, Systems Integration
Computing Review Category: H4.1, K6.5

1 Introduction

Separation of duty (SoD) is a security principle used to formulate multi-person control policies. In essence it stipulates that two or more different people must be responsible for the completion of a business process. It would thus, in principle, discourage fraud by requiring a conspiracy, thereby increasing the risk to the potential perpetrators [10].

Two basic variations of separation of duty exist, namely static separation of duty and dynamic separation of duty [10]. Static SoD requires that the membership to a set of roles must be strongly exclusive [7]. Business reality, however, may require the same user to belong to two roles in the set. This shortcoming of static SoD led to the introduction of various forms of dynamic SoD policies.

Dynamic SoD policies provide increased flexibility by controlling the activation and usage of roles. It thus removes the primary deficiency of static SoD by allowing users to act in multiple roles [12]. Dynamic SoD in its initial form simply prohibits a user to have more than one role active at any time. It, however, does not restrict the potential of a user to belong to another role. Several variations of dynamic SoD were proposed to meet some of the shortcomings of this basic form.

Object-based SoD allows a user to belong to multiple exclusive roles, but no user may act upon an object he has previously acted upon [5]. This strategy has the limitation of only allowing a user to act on a particular object once in a business process.

Operational SoD, on the other hand, allows a user to perform multiple actions while adopting multiple roles. However, this is only allowed if the union of the privileges of all roles with common users is not enough to complete an entire business process [3]. Operational SoD, furthermore, does not allow one user to perform all the actions in a business process, even if on different objects.

History-based SoD essentially combines the ideas behind object-based SoD and operational SoD. It allows one user to have multiple roles and also allows the union of the actions of those roles to span an entire business task. However, no user is allowed to perform all the actions in a business task on the same object. History-based SoD thus requires a detailed access history on each object [10].

The case study provided below will utilize various SoD policies to illustrate that a need for maintaining SoD policies across heterogeneous systems exists.

2 Business Scenario

The business scenario utilizes two heterogeneous workflow systems. The workflow systems are used by a motor vehicle manufacturing company to allow Dealers to claim back capital from the manufacturer’s Warranty Department for repairs done to cars that are still under warranty. Although closely linked in the business scenario the two workflow systems were developed independently.
over different time periods to fulfill a particular need within different sections of the Warranty Department. The business scenario utilizes a number of business roles and one object namely a claim form document. The roles of Warranty Consultant, Dealer and Warranty Supervisor are briefly defined. The Dealer role represents the entity that performs repairs on the vehicles. The Warranty Consultant role is filled by a person who processes the Dealer claims on the warranty workflow systems. Lastly, the Warranty Supervisor role is responsible for assisting the Warranty Consultants in making claims decisions in cases where the claims involve a large amount of money, and/or where there are irregularities in the claim circumstances.

The roles of Warranty Consultant and Warranty Supervisor form a hierarchy. The Warranty Supervisor is senior to that of Warranty Consultant. This structure implies that the Warranty Supervisor role can perform all the duties of the Warranty Consultant role as well as some additional duties that can only be performed by the Warranty Supervisor role.

The business scenario consists of two major business processes. A business process can be described as a set of activities with a common goal. They tend to be of long duration, possibly involving many users over distributed heterogeneous environments. Each business process consists of a number of tasks, each a step within the business process [1].

The two business processes involved in this case are the claiming process and the claim adjustment process. This paper will only focus on the claim adjustment process. However, understanding the basic business activities in the claiming process will enhance understanding of the claim adjustment process. A brief description of the business activities involved in the claiming process thus follows.

2.1 The Claiming Process

The Claiming Process is initiated when a Dealer claims money from the manufacturer for a warranty repair that he has successfully completed. The Dealer submits a warranty claim form to the manufacturer. The claim form is received by the manufacturer and assigned to a user fulfilling the Warranty Consultant role. This user processes the claim form utilizing the Claims Workflow System. Access to the claim form is exclusively through the Claims Workflow System. Therefore, when the claim form has been successfully processed, it is unavailable for further alteration. A claim form in this state is known as a closed claim. The claim form is then forwarded to the Payment Workflow System in order to arrange for the Dealer’s money to be electronically paid into his account.

Dealer claims are not paid immediately when received by the Payments system. A running total is kept of how much money each Dealer is owed. The Dealers are then paid the balance owed to them fortnightly.
2.2 The Claim Adjusting Process

There are certain conditions under which it may be necessary for a closed claim to be adjusted. For example, a Dealer may submit a warranty claim form with the incorrect part number or the incorrect number of parts. This claim is approved and closed by the Claims Workflow System and forwarded to the Payments Workflow System. However, it may happen that once the Dealer is reimbursed he notices that the amount paid to him is inconsistent with the amount he intended to claim. In order to correct this matter the Dealer must contact a Warranty Consultant and report the problem (see figure 1, activity 1).

The Warranty Consultant possesses the ability to adjust a closed claim under the supervision of a Warranty Supervisor. For any claim form to be adjusted the Warranty Consultant must in the first place recall the closed claim form on the Claim Workflow System (see figure 1, activity 2). Thereafter, in order to reopen the closed claim form, the Claim System will randomly select a second Warranty Consultant. He will be notified that a certain claim form needs to be reopened by a specific Warranty Consultant. The second Warranty Consultant must then confirm the reopening of the closed claim form before it can be processed (refer figure 1, activity 3), thus supporting the basic notion of SoD.

Once the closed claim form has been re-opened, the Warranty Consultant processing the claim form may make the necessary adjustments (see Figure 1, activity 4). Once this business activity is completed, the Claims Workflow System selects a Warranty Supervisor and forwards the adjusted claim form to him. This Warranty Supervisor must review the adjustments made to the claim form and authorize the Claims System to process the adjusted claim form (see figure 1, activity 5).

However, if the Warranty Supervisor rejects the adjustments made by the Warranty Consultant, he must provide a valid reason for the rejection. In such an event the details of the claim form are then added to an exception report for auditing purposes (see figure 1, activity 6). Thereafter the claim form, together with the reasons for rejection, are sent back to the Warranty Consultant who processed the claim for correction. Once the Warranty Consultant has corrected the claim form, it is again forwarded to the Warranty Supervisor who had rejected the claim for authorization (refer figure 1, activity 4).

Once the Warranty Supervisor authorizes the adjusted claim form, it is placed in a batch process to be run after normal business hours. The purpose of the batch process is to convert the documents processed by the Claim Workflow System for that day to a compatible document standard recognized by the Payments Workflow System.

When the Payments Workflow System receives the claim form documents, the documents are assigned to the various roles within the system for processing. All adjusted claim form documents are assigned to users fulfilling the Warranty Supervisor role (fig1, activity 7). They must ensure that the conversion from one system to the next was successful. They must also ensure that no abnormalities exist in the payment of the adjusted claim. It is important that the claim form is not assigned to the same Warranty Supervisor who authorized the adjustment of the claim in the Claims Workflow System. This is done to prevent a user from checking his own work and, accidentally or intentionally, allowing incorrect values. SoD is enforced through this as a "new" Warranty Supervisor become involved in the business process, thereby restricting the number of steps that any person may fulfill.

If the Warranty Supervisor does not accept the adjusted claim form in the Payment System, it is placed in a batch process to be returned to the Warranty Consultant who processed it in the Claims Workflow System. The Warranty Consultant must then re-start the adjusting process from activity 4, making the specified corrections. When the adjusting process again reaches activity 7 in the Payments System, the claim form must be sent to the same Warranty Supervisor who previously rejected the adjusted claim form in the Payment Workflow System.

When the adjustments made by the Warranty Consultant are authorized by a Warranty Supervisor in both the Claims Workflow System and the Payment Workflow System, the adjusted document is processed by the Payment System. The Dealer’s running total is then either debited or credited with the correct amount (figure 1, activity 8). Thereafter the authorized claim form is returned to the Claims Workflow System through the batch process and the authorized claim form is closed (figure 1, activity 9).

3 A need for history

In order for the Claims Workflow System to enforce the various SoD constraints discussed, it must support history-based SoD. This is because only history-based SoD will allow one user to have multiple roles, allow the union of the actions of those roles to span an entire business task and prevent the user from performing all the actions in the business process on the same object [10].

In order to utilize a history-based SoD policy the workflow systems need to know the specific history that a user has with an object. This history must be used to limit the activities that that user may perform on that object.

In a business process, utilizing only one workflow system, history could be stored in one central data repository. The workflow system could then use the data to enforce History-based SoD by checking whether a particular user performed a particular business activity on a particular object. It could then decide whether the specific user is eligible to perform the next business activity in the business process.

The problem with the above mentioned business scenario is that it contains two heterogeneous workflow systems that need to share the same history information. The problem of heterogeneity therefore needs to be addressed, specifically in terms of how the history related
information of a particular document is to be shared between the two separate incompatible workflow systems.

To solve this problem a technology must be utilized that allows the claim form document’s history information to be passed around with the document in the heterogeneous workflow environment in a platform independent manner. XML promises to do this.

4 XML

XML is a declarative, tag-based mark-up language for describing the structure and contents of a document [2,9]. XML is all about describing information in such a way that it can be understood by different applications, thereby separating the user interface from the data [6].

Both HTML and XML are derived from the ISO Standard Generalized Markup Language (SGML). XML is a slimmed down Web Based subset of SGML [2,6]. HTML on the one hand is an application of SGML, meaning that its tag semantics and tag set are fixed. XML on the other hand provides a facility to define tags and the structural relationship between them. Since there is no predefined tag set, there cannot be any preconceived semantics. All of the documents semantics are thus defined by the applications that use it [11].

Each document can contain a set of rules called a Document Type Definition (DTD). The DTD defines the document’s structure allowing any application using XML to verify that the document is authored according to a predefined structure [4]. This allows other authors to easily create the same class of document, by following the structure laid out in the DTD [6].

All XML coding and tagging are done by utilizing standard text formats, allowing data in an XML document to be easily converted and read by any system or application that supports XML [6]. A variety of XML parsers and verifiers are available from vendors allowing for the easy development of custom validation and conversion programs [11].

XML has already made its mark in the workflow arena as a problem solver for heterogeneous environments. The Workflow Management Coalition (WMC) has produced a draft specification, which was released for industry comment in June 1999. This specification builds on the WMC’s earlier work by evolving its existing workflow interoperability standards into XML-based exchanges between workflow systems. The new standard is known as Wf-XML and contains a definition of the basic DTDs defining the XML encoding of workflow messages to support interoperability. It is the WfMC’s intention to extend the standard to include workflow operations from the other WMC interfaces in order to form a complete XML-based specification for all workflow functions [8].

In contrast to the Wf-XML standard which uses XML to encode messages between workflow systems to support interoperability, this paper wishes to define an XML document which travels with workflow objects storing history data as it moves along.

5 Proposed Solution

This paper proposes that the information needed to enforce history-based SoD travels in XML format together with the warranty claim form document from one business activity to the next gathering history information or “baggage”. This solution is not concerned with the storage of this "baggage" document as a variety of techniques could be used.

The baggage will be maintained and interpreted in a three step process, namely baggage evaluation, document processing and baggage collection.

Figure 2 utilizes business activities 4 and 5 from Figure 1 to show that each business activity is preempted with a baggage evaluation step, followed by the normal document-processing step discussed in the business scenario, and lastly the baggage collection step.

When encountering a new task, the workflow system must determine whether any SoD rules exist for the document being processed for the current business activity. If such a rule does exist, the workflow system evaluates the baggage section of the claim form to identify all the users who have previously performed actions on the document (object). A list of users able to perform the business activity without violating SoD rules is then computed. Thereafter the work items are placed on the respective work lists.

The document-processing step follows the baggage evaluation step. It involves the normal document processing, as described in the business scenario and illustrated in Figure 1, to be done by the user selected in the baggage evaluation step.

The last step to be performed in the business activity is the baggage collection step. In the baggage collection step
the history information, or baggage, of the business activity that has been completed by the user is added to the existing baggage of the document. The document is then transferred back to the workflow system from where it can progress to the next business activity. The workflow system uses the updated document baggage in the baggage evaluation step of the next business activity to enforce the SoD rules.

Now consider the expression of the separation of duty rules.

6 XML Separation of Duty Rules

Key to the enforcement of the SoD rules utilizing XML is the expression of the SoD rules. Figure 3 provides an excerpt from an XML SoD policy document. This document is used to compute the list of users able to perform a particular business activity without violating any SoD rules.

The SoD policy document consists of a number of entries called rule_entry sections that can occur multiple times within the SoD policy document. The SoD policy document will be searched to determine if any SoD rule entries exist for the business activity being processed. Each rule_entry section is divided into 3 subsections, the target_elements section, the rule section and the conditional_elements section.

The target_elements section is responsible for storing the information that allows the workflow system to identify the business activity that the rule pertains to. The first three data fields of the target_elements section are the target_workflow, target_process and the target_activity data fields. In figure 3 we can see that the rule_entry refers to business activity “activity_7”, business process “claims_adjustment” of the “payments_system” workflow system.

The target_object data field identifies the type of object that the SoD rule applies to. In the case of figure 3 the SoD rule applies to all “claim_form” object. The last data field in the target_elements section is the target_type data field that identifies the property of the workflow object that the SoD rule refers to. According to figure 3 this is the “user” property of the “claim_form” workflow object.

The rule section is a stand-alone data field that identifies the relationship between the target_elements and the conditional_elements sections. This relationship can either be “equal” or “not equal”. In figure 3 we can see that the relationship between the target_elements and the conditional_elements is “not equal”.

The conditional_elements section stores all the information needed to uniquely identify the business activity in the document’s baggage, which contains the SoD information needed to enforce a SoD rule. The conditional_elements section contains three data fields, namely the conditional_workflow, conditional_process, and the conditional_activity fields. In figure 3 these fields refer to “activity_5” in the “claim_adjustment” business process on the “claims_system” workflow system. The combination of these three data fields allows the workflow engine to identify the SoD entry in the document’s baggage that contains the information needed to enforce the SoD rule.

Now let us consider how the SoD policy document described in this section can be used to provide history-based separation of duty in a heterogeneous workflow environment.

7 XML Baggage

At the heart of the solution described by this paper lies the XML baggage structure. The structure of the XML baggage will go a long way to determining the flexibility, adaptability and the integrity of the entire workflow security system. Figure 4 provides an extract of an XML baggage document used to store the baggage for the claim form document of the example business scenario discussed previously.

The XML baggage document consists of a number of entries called history_entry sections that can occur multiple times within the baggage document. Every time the claim form document participates in a new business activity, a new history_entry is created to store the history data for that particular activity. The history_entry section is divided into 2 subsections, namely the environment_section and the user_section.

The environment_section stores all the data pertaining to where the business activity was performed. This “where” does not refer to the physical location of the business activity, but rather to the workflow system. The environment_section is made up of three data fields, the first of these being the workflow_system data field. The workflow_system data field reflects the name of the workflow system on which the business activity occurred. The next data field in the environment_section is the
<history_entry>
  <environment_section>
    <workflow_system>claims_system</workflow_system>
    <business_process>claim_adjustment</business_process>
    <business_activity>activity_5</business_activity>
  </environment_section>
  <user_section>
    <user_id>carl1</user_id>
    <user_role>warranty_supervisor</user_role>
  </user_section>
</history_entry>

The **user_section** element stores all the information needed to uniquely identify the user that performed the business activity. The first of the **user_section**’s two data fields is the **user_ID field**, which stores the unique user ID of the user who performed the business activity captured in the business activities **environment_section**. The second data field in the **user_section** is the **user_role field**. The **user_role field** stores the role assumed by the user while performing a particular business activity. It is important to capture this information since a user may be capable of fulfilling multiple workflow roles within a single business process. The role selected may then affect his ability to perform other business activities later on in the business process.

Now that the history information to be stored by the document has been discussed in detail, an application example is provided below, demonstrating how this technique would be used to enforce history-based separation of duty policies in the business scenario.

### 8 Application

In order to illustrate how XML baggage can be used to aid in the enforcement of History-based Separation of Duty in a heterogeneous workflow environment we must once more examine the business scenario discussed in section 2 of this document and illustrated in Figure 1. In particular we will be examining the link between business activities 5 and 7 as illustrated in Figure 5. This link is activated when the Claims Workflow System’s Warranty Supervisor is satisfied with the adjustments made by the Warranty Consultant in business activity 4. The claim is then submitted to the Payments Workflow System for further processing. These two business activities have been chosen because they clearly demonstrate History-based Separation of Duty between two separate workflow systems.

In order to illustrate how History-based SoD utilizing XML documents can be applied to complete this workflow step, we will assume that the claim form has been accepted in activity 5, and has been submitted to the claims workflow system to be processed by activity 7.
8.1 Baggage collection

Before the claim form can be passed from activity 5 to activity 7 the baggage collection step of activity 5, which is highlighted in Figure 5, must be completed. As discussed previously in section 6 of this document the baggage collection step gathers the history information, or baggage, of the business activity that has just been completed, and stores it with the claim form’s existing baggage in its XML baggage document. The history information stored in the XML baggage document includes the workflow system, business process and business activity names as well as the user name and role of the user that performed the activity. Once this step is completed, the document is passed back to the workflow system so that it can progress to business activity 7. The collected baggage for business activity 5 is shown in Figure 4.

8.2 Baggage evaluation

Concurrently to receiving the claim form, the Payments Workflow System is sent a message by the Claims Workflow System requesting the initiation of business activity 7. In order for this to happen, the claim form must be assigned to a user.

When the Payments Workflow System wishes to assign a claim form to a user, it must first create a user list of legitimate users to pick from (Figure 6 step 1). The user list is created by having the payments workflow system reference its internal tables to determine which business roles may perform business activity 7. In this case the Warranty Supervisor role can perform business activity 7. Next, the Payment Workflow System constructs the user list by selecting all the users that may fulfill the Warranty Supervisor role.

Once the user list has been compiled, the workflow system must determine if any separation of duty constraints exist for business activity 7 by searching the constraint_entry sections of the SoD rules XML document for possible matches (Figure 6 step 2). The search will present a match for business activity 7 if any entries are found where the target_workflow, target_process, target_activity and the target_object data fields contain the entries, ‘payments system’, ‘adjustment process’, ‘activity 7’ and ‘claim form’ respectively.

If no matches are found in the SoD rule XML document the workflow system accepts that no SoD constraints exist for the business activity in question. The claim form can be assigned to any user on the user list (Figure 6 step 3).

In the case of business activity 7 however, one match is found, which is illustrated in Figure 2. Next the workflow system must determine the type of SoD constraint and rule that must be applied (Figure 6 step 4). To achieve this the workflow system must examine the target_type data field and the rule data field of the matched constraint_entry. The target_type data field will reveal that the SoD constraint restricts the users that may perform business activity 7. The rule data field will reveal that the user that must perform activity 7 may not equal the same user that performed the business activity in the conditional_elements section. By querying the conditional_elements section of the constraints_entry the entire SoD constraint can be understood. It states that the user who originally performed business activity 5 may not perform business activity 7.

The next step in enforcing business activity 7’s SoD constraint is to search the claim forms XML baggage document in order to determine the user name of the user that originally performed activity 5 (Figure 6 step 5). This is done by searching the baggage document until a history_entry is found where the data in the Workflow_System, Business_Process and Business_Activity data fields match that of business activity 7’s. This history_entry is illustrated in Figure 4. When the history_entry is found the workflow system reads the user_Id data field from the history_entry’s user_section to determine the user name of the user that originally performed business activity 5.

Once the user name has been found the workflow system must remove all names from the user list that match that of the user name in the user_section (Figure 6 step 6). The user names remaining on the user list are those of the users that can legally process the claim form in business activity 7 without violating any of the SoD

Figure 6. The baggage evaluation step
Once all of these steps are completed the workflow system can assign the claim form to any of the remaining user names on the user list for processing (Figure 6 step 3). Once the document has been processed by the user the baggage collection step must again be completed before the workflow system can perform the next business activity.

9 Conclusion

In order to demonstrate how the XML baggage approach can be used to enforce History-based Separation of Duty policies across heterogeneous workflow environments, it was necessary for the authors to describe a ‘perfect world’ approach. Now consider some of the issues that were not discussed or elaborated on due to space limitations or their relevance to the message that this document was attempting to convey.

An issue to ponder is the size of the baggage. A document that passes through several large and complicated business processes, may quickly build up baggage that can be deemed excessive. Ways must be explored to limit the size of XML baggage by only maintaining baggage that is relevant to future business activities, thus optimizing the stored baggage.

Separation of duty is an information security concept. However, utilizing text documents presents security threats. The integration of other security mechanisms into the approach will need consideration to ensure the protection of information. In the longer run, it is foreseen that the baggage should become part of a document, thereby allowing the document to have autonomy as far as access control decisions is concerned.

This paper, however, shows that XML is indeed a flexible data representation mechanism which can be used with great effect in heterogeneous workflow environments.

References


