Abstract

A workflow process involves the execution of a set of related activities over time to perform a specific task. Security requires that such activities may only be performed by authorized subjects. In order to enforce such requirements, access to the underlying data objects has to be controlled. We refer to such access control as level 1 access control. In addition, when an individual is authorized to perform an activity, access should be limited to the time that the activity is being performed: Access to activity information before an activity commences or after it has terminated may be undesirable. This we will refer to as level 2 security. Finally, applications often specify application-oriented (level 3) security requirements. This paper considers security restrictions in the latter category and proposes a rigorous approach that may be used to specify such policies. Enforcement (implementation) of such policies is also considered. The paper assumes that level 1 and level 2 mechanisms are in place and builds level 3 security mechanisms on these underlying levels.

1 Introduction

The formal specification of authorisation in workflow systems has received some attention [2]. However, an approach to precisely specify the security requirements of a specific workflow application is still required. For example, it is necessary to be able to precisely specify the required relationship between actors in a workflow system (such as ‘only the manager of an employee may approve leave applied for by the employee’). In addition it should be possible to precisely express other security notions such as separation of duties or application-specific restrictions. This paper forms part of work to provide an environment where such requirements may indeed be specified precisely.

In order to do this, a workflow system should be viewed on three distinct levels: On level 3 the workflow application operates, be it an order, claim, or any other specific application. It is natural to phrase the level 3 requirements in terms of ‘primitive’ workflow notions such as activities. In fact, many security requirements are common to all workflow applications, such as the fact that users should only be granted access to objects while they require the access to perform some activity; these aspects are viewed as level 2 issues. Note, in particular, that level 2 forms the environment in which the level 3 specifications will be embedded.

Many workflow systems operate over a variety of underlying platforms; in particular it is necessary to store the data associated with the workflow processes somewhere. In this paper we assume that level 1 consists of (one or more) databases where the information is stored.

This paper is structured as follows: Section 2 contains background information on workflow management systems and on security. Section 3 contains a running example that will be used through the remainder of the paper. Section 4 gives the necessary background about security issues on levels 1 and 2. Section 5 illustrates how the environment created on the lower levels may be used to specify application-level access restrictions. Section 6 compares our approach to existing work. Section 7 concludes the paper.

2 Background

A workflow process or activity network is a set of activities that are connected to achieve a common goal [13]. It shows how these various activities are arranged to achieve the goal.
Use of the workflow system causes the creation of workflow process instances and activity instances. Each such instance is responsible for handling a particular ‘case’ of a workflow process or activity [12]. Each instance may have its own local data [13]. In practice each instance may operate on its own data as well as on global data. A claim that is submitted and processed will be unique to a particular workflow process instance. In contrast, current amounts payable for a given type of claim will be global in the sense that the same data will be shared by all claim instances. We assume that all data in the underlying system is represented as objects.

Workflow process models are often based on extended versions of Petri-nets [1, 2]. Our model is also essentially based on this notion, but uses a notation that enables us to consider some aspects of process activities in more detail.

3 Running example

This section contains a simple example of a workflow process definition that will be used to illustrate aspects in the remainder of this paper. The example is the familiar one of claim submission, approval and settlement and is depicted in figure 1.

![Diagram](image)

**Figure 1. The example workflow process definition**

Some remarks are in order: A process consists of activities that follow one another in some structured way; in the diagram activities are indicated by ovals: Submission, Approval and PayOut represent the logical sequential steps in the simple claim handling process; the intuitive meaning of the names are adequate for the moment. StatusCheck is an activity that may be performed by the various (authorised) participants in the workflow process to check the current status of the process.

Activities are connected by transitions; in the diagram transitions are indicated by solid dots. Where beneficial, transitions have been named. Note that some ‘transitions’ represent ‘final states’ since they do not serve as input to subsequent activities. This is due to an assumption that we make that an activity terminates when the ‘token’ is passed from the activity to the transition. Since we assume that when a workflow aborts, or PayOut is finalised, that the preceding activity effectively terminates, transitions are provided to enable the activities to terminate. From an authorisation viewpoint, in particular, is it important to terminate (some) activities since termination coincides with revocation of access rights. Workflow processes are, on the other hand, often initiated by performing an action. A user who initiates the Submission activity may do it by starting to complete a claim form. The Submission activity therefore forms an initial activity of the workflow process. It is also possible that a workflow process may be initiated by ‘receiving a token’ from outside the workflow system. In such a case the workflow process may be initiated from an initial transition rather than an initial activity.

4 Levels 1 (database) and 2 (workflow) security

In other work [10] we are busy specifying the structure of a generic workflow process in Z (see [4] for an introduction to Z). This will enable us to precisely specify the generic security requirements of workflow systems and thereby guide the implementation of secure workflow management systems. Remember that level 1 concerns the objects to which access needs to be regulated, while level 2 groups such object accesses together into activities. This enables one to specify access restrictions on level 3 in terms of the natural notion of activities. To illustrate, the activities used in the example from section 3, may be specified as follows:

\[ ACT \ ::= \ Submission \mid Approval \mid PayOut \mid StatusCheck \]

On level 2 the precise list of activities is not important; it is only important that some such list will be defined on level 3.

The (workflow) process description graph gives a static description of the workflow process; in operation multiple process instances may be active at any given moment — all using the underlying workflow process definition. It is therefore also necessary to be able to identify the process instance, using some process instance identifier:

\[ \text{PROC} \]

A workflow process instance therefore consists of a sequence of (possibly overlapping) activity instances. At a time \( t \) process \( p_1 \) may be at the point where John is busy submitting a claim, process \( p_2 \) where Jane is approving it, etc. Note that more than one activity instance of some process instance may be active simultaneously (whenever parallel activities occur in the process description). For simplicity we will assume that only a single instance of an activity may occur during the lifetime of a process instance. This will allow us to denote an activity instance by referring
to the process instance identifier (PROC) and the activity description identifier (ACT). Where this is not sufficient it will be necessary to further identify the activity instance by some occurrence counter. Although this is a rather simple extension, it is not required for the purposes of the current paper.

5 Level 3 (application-level) security

The contribution of this paper is the consideration of formal specification of application-level security policies for workflow systems. Given the infrastructure described above, it is now possible to specify security policies for workflow systems formally. Three alternative mechanisms seem particularly appropriate for application-level (level 3) security: a dynamic access control matrix, an unacceptable history approach and a dynamic prohibition approach. These approaches are introduced below and discussed in more detail in sections 5.1, 5.2 and 5.3.

A dynamic access control matrix (DPCM) is similar to an ordinary access control matrix: it contains subjects and objects and the corresponding allowed access modes that each subject has to each object. The major difference between DPCM and traditional access control matrices is the fact that an entry in the DPCM does not imply that the subject may access the object, but only that the subject may be granted access.

The unacceptable history approach assumes that a log is kept of events that occur in a workflow system. The unacceptable history approach then specifies security policies in terms of (combinations of) entries that are not allowed to occur in these ‘history’ logs. The unacceptable history approach provides a concise approach for specifying application-level security policies.

The dynamic prohibitions approach is similar to the dynamic access control matrix approach in that a set of prohibitions (negative authorisations) are dynamically modified as specific workflow process instances progress. Further, these sets identify specific activity instances to which the identified subjects are denied access. Dynamic prohibitions differs from the DPCM approach in that a set of positive authorisations is also specified, but in terms of process descriptions. The effective access rights are then determined by the ‘difference’ between the positive and the negative set. More details about this approach are given below in section 5.3.

The DPCM approach, while intuitive, suffers from two major drawbacks: excessive run-time overhead and security specifications that are procedural in nature. The unacceptable history approach enables one to concisely specify policies, but implementation of such policies is, unfortunately, not straightforward. Dynamic prohibitions seems to have a much lower run-time overhead than the DPCM approach, but are also procedural.

Given the relative advantages and disadvantages of the these approaches, the following approach is recommended: Specify the policy using the concise unacceptable history approach. Then convert the policy to dynamic prohibitions. It seems possible to automate this conversion. The implementation is then based on the low run-time overhead dynamic prohibitions approach. Space restrictions do not allow discussion of this recommendation in the current paper.

5.1 Dynamic access control matrix

Entries in the DPCM may be viewed as triples \((s, p, a)\) where \(a\) indicates the activity that subject \(s\) has been, in principle, granted authorisation to perform; \(p\) identifies the process instance (and \((p, a)\) therefore indicates the activity instance).

\[
dpcm : \mathbb{P}(\text{SUBJ} \times \text{PROC} \times \text{ACT})
\]

The phrase in principle above indicates that \(s\) will not necessarily be granted authorisation to actually perform the activity. In our formalisation of levels 1 and 2 we have assumed that a single subject is always exclusively responsible to perform any specific activity; where more than one subject has to cooperate to perform a ‘activity’ that activity’ may be split into (sub-)activities, each performed by a single subject. If managers are authorised (in principle) to perform claim approval, entries will be made in the DPCM authorising all managers to, in principle, perform the claim approval activity of every process instance \(p\). Note that only a single manager will eventually actually be selected from those who are authorised in principle to perform the activity. See [10] for a discussion of how the actual subject may be selected from the potential subjects. One simple example that is sufficient for the purposes of the current paper is to visualise a task list listing the available activities for all those subjects who are authorised in principle to perform them. When one of these subjects \(s\) selects this activity, \(s\) is actually authorised and the activity is removed from all other subjects’ task lists.

Suppose that

\[
\begin{align*}
&\text{emp} : \mathbb{P} \text{SUBJ} \\
&\forall s : \text{SUBJ} : p : \text{PROC} \bullet \ (s, p, \text{Submission}) \in \text{dpcm} \Rightarrow s \in \text{emp}
\end{align*}
\]

is a set of employees and that only employees are allowed to submit claims. This is easily expressed as:

\[
\forall s : \text{SUBJ} : p : \text{PROC} \bullet (s, p, \text{Submission}) \in \text{dpcm} \Rightarrow s \in \text{emp}
\]

Similarly, if only managers

\[
\begin{align*}
&\text{managers} : \mathbb{P} \text{SUBJ} \\
&\text{managers} \subseteq \text{emp}
\end{align*}
\]
are allowed to approve claims, this may be expressed as:

\[
\forall s : \text{SUBJ}; p : \text{PROC} \bullet
(s, p, \text{Approval}) \in \text{dacm} \Rightarrow s \in \text{managers}
\]

Note that there is no need to delay adding triples to \text{dacm} to the point where the \text{Approval} activity is to be performed. The inherent mechanisms in level 2 will ensure that no subject is actually authorised to perform an activity until the activity becomes enabled (ie, all preceding activities have completed).

When using the DACM approach the intention is that all activity-role assignments are initially made (quantified over all process instances) similar to the examples above. Below we will refer to such assignments as general policies.

In [10] a Grant procedure has been defined that grants access to an appropriate subject when required by the workflow process. It is obviously simple to extend that procedure such that it only grants access to subjects listed in \text{dacm}.

In addition to the general policies discussed above, workflow systems inherently include ‘dynamic’ policy elements: the classical example here is that a manager who submits a claim should not be allowed to approve the same claim. What this means in practice is that performance of any activity may lead to the removal of authorisations from \text{dacm} — hence the term dynamic in the name of the approach. For example, if subject \( s \) performs a Submission activity during process \( p \), the policy may state that

\[
\text{dacm}' = \text{dacm} \setminus (s, p, \text{Approval})
\]

(Note that, if \( s \) is not a manager, the specification above will leave \text{dacm} unchanged.) Below we will refer to specifications that change dynamically because of activities that are executed, as specific policies.

As indicated earlier, the DACM approach suffers from a number of drawbacks. Firstly, if the \text{dacm} set is directly implemented, every general policy will lead to a significant number of entries in \text{dacm}. The run-time storage requirements of \text{dacm} may therefore be excessive. In addition, if a policy consists of a series of modifications to a set the intention (or correctness of portrayal of the intention) may be difficult to establish. However, since the following two approaches are based on the DACM approach, understanding the DACM approach is necessary.

The next approach to be described is the unacceptable history approach. Its primary difference from the DACM approach is the fact that it is declarative rather than procedural.

### 5.2 Unacceptable history approach

The second mechanism that will be useful to express policies is based on a history of actual authorisations:

\[
\text{history} : \text{bag} (\text{SUBJ} \times \text{PROC} \times \text{ACT})
\]

It is now simple to modify Grant to add any authorisation triple it deals with, to \text{history}.

In this case general authorisations are given as illustrated by the following example:

\[
\forall s : \text{SUBJ}; p : \text{PROC} \bullet
(s, p, \text{Submission}) \in \text{history} \Rightarrow s \in \text{emp}
\]

where \text{emp} is again a set of all employees. General policies here are characterised by the fact that they are quantified over all subjects and processes and place restrictions on the subjects.

The name of this approach was selected from the mechanism used to specify specific policies. In general the occurrence of one event in \text{history} prohibits other events. To illustrate,

\[
\forall s : \text{SUBJ}; p : \text{PROC} \bullet
(s, p, \text{Submission}) \in \text{history} \Rightarrow
\neg (s, p, \text{Approval}) \in \text{history}
\]

specifies that the submitter of a claim cannot approve it.

If a \text{manager} function is defined,

\[
\text{manager} : \text{SUBJ} \rightarrow \text{SUBJ}
\]

the following policy specifies that a claim has to be approved by a manager of the claimant.

\[
\forall s : \text{SUBJ}; p : \text{PROC} \bullet
\text{manager}(s), p, \text{Approval} \in \text{history} \Rightarrow
\neg (s, p, \text{Submission}) \in \text{history}
\]

If a policy requires that the same subject may not be responsible for any two or more activities in any given process instance, it may be specified as follows:

\[
\forall s : \text{SUBJ}; p : \text{PROC}; a, b : \text{ACT} \bullet
(s, p, a) \in \text{history} \land a \neq b \Rightarrow \neg (s, p, b) \in \text{history}
\]

An alternative policy may require that two subjects cannot mutually approve one another’s claims:

\[
\forall s, t : \text{SUBJ}; p, q : \text{PROC} \bullet
\[(s, p, \text{Submission}), (t, p, \text{Approval}),
(t, q, \text{Submission}) \in \text{history}
\Rightarrow \neg (s, q, \text{Approval}) \in \text{history}
\]

It is obvious from the examples above that the unacceptable history approach is both powerful and concise. It is, unfortunately, hard to implement directly since a direct implementation will only detect problems once they have already occurred, ie, once they are in the history list. The unacceptable history approach is, fortunately, simple to map to the following approach that does not suffer from this drawback.
5.3 Dynamic prohibitions approach

Above a set of positive authorisations was used to specify the potential performers of activities; a history set was used to prevent an ‘invalid history’ from occurring. An interesting combination is worth investigation: Let

\[
\text{positive} : \mathcal{P}(\text{SUBJ} \times \text{ACT})
\]

contain the general policies. (Here general policies are assumed to be similar to those used in the DACM approach above; there such policies were quantified over all processes, which means we do not lose any generality with this view. positive is, however, a significantly smaller set than \textit{dacm}.) Note that positive is intended to be a static set that is not modified over time.

Let

\[
\text{negative} : \mathcal{P}(\text{SUBJ} \times \text{PROC} \times \text{ACT})
\]

be a set where \((s, p, a) \in \text{negative}\) indicates that subject \(s\) is not allowed to perform activity \(a\) of process \(p\) — overriding any positive authorisations that may exist. In other words, the set of potential accessors is restricted to

\[
\text{potential} = \{ \forall p : \text{PROC}; s : \text{SUBJ}; a : \text{ACT} \mid (s, a) \in \text{positive} \} \setminus \text{negative}
\]

Like \textit{dacm} and \textit{history}, negative is inherently dynamic: whenever \textit{history} changes in such a way that some triple \((s, p, a)\) is no longer allowed to occur in \textit{history}, that triple is simply added to negative. \textit{Grant} may then enforce the policy similar to enforcement in section 5.1.

Since an action generally excludes some specific participants from some specific activities in future, the set negative is expected to be rather small — much smaller than \textit{dacm}. And above it has been argued that positive is indeed much smaller than \textit{dacm}. It is our contention that the space requirements posed by negative and positive combined are not only much smaller than that of \textit{dacm}, but realistic to implement directly.

Note that negative authorisations have been used in various places in the literature — most notably [11]. It is possible to use negative in a similar way in the current approach. However, until provision is made for hierarchies of tasks (in other words, task sub- and superclasses) the benefits (and complexities) originating from negative authorisations in those cases do not hold for the current approach.

6 Comparison with related work

The only other paper that we are aware of that deals with the issues addressed in this paper is [3]. Where the current paper only focusses on application-level security specifications, [3] also deals with issues such as constraint consistency and role assignment. That paper also assumes that tasks are not uniquely assigned to roles (as we do), so that restrictions on the roles that may perform an activity also become an issue (in addition to restrictions on the individual subject who may perform an activity). While those issues are indeed relevant, we have decided to only deal with restrictions on individual subjects in the current paper in order to focus specifically on such constraints in a simpler environment. In addition, where [3] also discusses static constraints on the assignment of roles and individuals, we only focus on dynamic constraints.

However, as far as the central theme of the current paper is concerned, we contend that our notation for specifying policies is more concise. In addition, the approach specified in [3] is unclear on whether some restrictions apply to some workflow process instance, or all instances in general: for example, their constraint that the same role must execute tasks \(T_1\) and \(T_2\) seems to be implemented as “if, in instance \(k\), role \(r\) executes task \(T_1\), then \(r\) must execute all future instances of \(T_2\)” While this problem may be easily remedied, the approach still makes it difficult to specify constraints such as “two managers are not allowed to mutually approve one another’s claims.” Furthermore, the specification of constraints such as “a subject is not allowed to perform more than one activity in a given workflow process instance” is simple to specify using our notation, but requires a lengthy specification in their notation. In particular, if an activity is added to or dropped from the workflow specification, their specification will have to be changed to reflect the change, while ours need not be changed.

One potential approach is to investigate the automated translation of our notation to that used in [3]. That will address our criticisms of lengthy and process dependent specifications and make the benefits of their approach available to our work.

Also see [3] for a list of other relevant research on constraining subject assignment as well as reasons why the other work do not address the new issues introduced by workflow systems. Note, in particular, that a history concept has been investigated in the context of flow control for a long time; see [5, 8, 7] for a discussion of these issues in object-oriented systems. However, note that \textit{history} in information flow addresses different issues from those addressed by \textit{history} in workflow systems.

7 Conclusion

This paper considered the specification of application-level security in workflow systems. Three mechanisms were considered: a dynamic access control matrix, an unacceptable history approach and a dynamic prohibition approach.
A dynamic access control matrix (DACM) is an intuitive, but impractical approach. The unacceptable history approach provides a concise, declarative notation for specifying application-level security policies. Implementation of such policies is, unfortunately, not straightforward, but this is addressed by the dynamic prohibitions approach since the latter avoids the overhead of the DACM approach, is procedural and therefore easily implemented and may be automatically derived from an unacceptable history specification.

Research that remains to be done is to embed the approach recommended in this paper in a specific workflow product to prove the concept in practice.

References


