Refactoring service-based systems: how to avoid trusting a workflow service

Howard Chivers∗,† and John McDermid

Department of Computer Science, University of York, Heslington, York YO10 5DD, U.K.

SUMMARY

Grid systems span multiple organizations, so their workflow processes have security requirements, such as restricting access to data or ensuring that process constraints are observed. These requirements are often managed by the workflow component, because of the close association between this sub-system and the processes it enacts. However, high-quality security mechanisms and complex functionality are difficult to combine, so designers and users of workflow systems are faced with a tradeoff between security and functionality, which is unlikely to provide confidence in the security implementation. This paper resolves that tension by showing that process security can be enforced outside the workflow component. Separating security and process functionality in this way improves the quality of security protection, because it is implemented by standard system mechanisms; it also allows the workflow component to be deployed as a standard service, rather than a privileged system component. To make this change of design philosophy accessible outside the security community it is documented as a collection of refactorings, which include problem templates that identify suspect design practice, and target patterns that provide solutions. Worked examples show that these patterns can be used in practice to implement practical applications, with both traditional workflow security concerns, and Grid requirements. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: workflow; security; Grid; service-based architectures; Web services; refactoring; pattern

1. INTRODUCTION

The need for security depends on the risk profile of a system, derived from stakeholders’ concerns for their assets, and an assessment of likely threats and known vulnerabilities. Although the risk to many grid science systems is only moderate, and can be dealt with by campus protection and recovery

∗Correspondence to: Howard Chivers, Department of Computer Science, University of York, Heslington, York YO10 5DD, U.K.
†E-mail: hrchivers@iee.org

Contract/grant sponsor: Royal Academy of Engineering

Copyright © 2005 John Wiley & Sons, Ltd.
mechanisms, projects with industrial, biomedical, or safety content, or that utilize high-value resources, are likely to have security concerns that demand confidence in their design and implementation. Without this confidence the operational deployment of the system may be prejudiced.

The approach described in this paper was developed to support the DAME Grid application [1], which includes a business workflow that spans several companies. DAME has commercial security requirements, and the security risk analysis of the system indicates that it is not possible to implement them with a centralized enactment system.

The close relationship between business processes and the workflows that implement them result in systems that try to accommodate process security in the workflow planning and enactment components. Mixing complex functionality and security is likely to result in the worst of both worlds: security overheads may impede functional performance, and complexity is the enemy of security assurance. In Grid and similar service-based systems, the situation is worse: the defining feature of a Grid is that it spans multiple organizations; these may have conflicting security requirements, or require that some security guarantees are independent of their collaborators. Centralized security protection and record keeping may simply not be acceptable.

This paper proposes that process security should be enforced by standard distributed-system protection mechanisms, as far as possible separating security functions from those that enact the workflow. This avoids the need for workflow sub-systems to be both highly functional and also security critical, and naturally distributes process security policies into the trust management framework of the distributed system.

It is important to distinguish between function and guarantee. Reducing the need for confidence, or trust, in the workflow component does not change its function; constraints that are enforced outside the workflow must still be observed by the plan if it is to be successfully enacted. What is removed is the need to guarantee the behaviour of this component.

This paper documents appropriate security design practice as a series of refactorings. Each refactoring includes a Problem Template, which identifies a system configuration with a security problem. The problem is resolved by a design pattern that preserves the functionality of the system while improving its security, and a description of how the pattern relates to workflow-based systems. The final section provides two worked examples and a discussion of implementation issues. The examples show how these patterns can be used to implement realistic processes: a payment workflow, and a formal collaboration. The former includes separation of duty, which is a common workflow security requirement; the latter shows a process shared between two different organizations, which is typical of information-based Grid systems.

These are system-level designs: processes or applications are described in terms of services and the data items they exchange. From this service-based architecture viewpoint the workflow sub-system should also be a service (e.g. a Web service [2,3]). The separation of security and functionality proposed in this paper facilitates this design philosophy by removing the need for workflow sub-systems to be deployed as privileged system components.

The security principles that motivate this work (see Section 2) are not novel and examples of their application can be found in well-designed infrastructure. However, we are not aware of previous work that applies security design practice systematically to systems that incorporate workflow, or that shows how to partition process enactment and security. These results can be applied by workflow architects to structure their security mechanisms, or by system designers who need to incorporate workflow services.

The contribution of this paper is as follows.
To show that security requirements previously incorporated in workflow enactment systems can be better supported by standard distributed-system protection mechanisms outside the workflow component. This separates security and functionality, allowing security requirements to be upheld by well-understood mechanisms, and facilitating the deployment of workflow as a service, rather than a privileged system component.

To provide recognizable examples of security design problems in service-based systems, and to give patterns for how they can be corrected. This allows non-security specialists to make use of the design experience.

The next section briefly describes the background to workflow security, security principles, and refactoring. Section 3 describes typical problems, and explains how they can be recognised and refactored. Section 4 presents worked examples of how these patterns can be used in practice, followed by a discussion of implementation issues. Section 5 concludes the paper.

2. BACKGROUND: WORKFLOW AND SECURITY

Terminology. A Workflow process, or just process, describes the requirement for a managed and structured flow of work and documents; a workflow sub-system, or component, implements the planning and enactment of a process. A workflow service is a workflow component deployed as a standard service in a service-based or grid architecture.

In addition to planning and enactment processes, workflow sub-systems provide a range of supporting functions [4], including user interface management, application launching, and job monitoring. Workflow security is concerned with these peripheral functions, as well as enactment, so it includes requirements that apply to the data and services associated with the process. This scope results in a wide range of potential security requirements [5–8], but the most important are process integrity (e.g. supporting separation of duty), data provenance, and data confidentiality.

The most common approach to process integrity is constrained planning [9–11]. Confidentiality of peripheral data is often supported by integrating role-based access control in the enactment system [12–14]. These approaches require a workflow system to deliver extensive security guarantees, as well as perform planning and enactment functions, and this is evident in systems designed to incorporate security [15,16].

As noted in the introduction, mixing complex functionality and security is likely to result in the worst of both worlds, and there are types of Grid system in which centralized security functions of this sort are simply not acceptable. However, the function of a workflow system is no more complex than an operating system, and over time designers have learned how to factor the security in such systems into well-understood and relatively limited areas of functionality. This is the topic of this paper: how to factor apart the security and functionality requirements of processes in order to comply with established security principles, and allow the distribution of security functions between collaborating organizations.

This amounts to refactoring the usual system design: modifying the structure of data and services so that the functionality is maintained, but security is enhanced. This is usually applied to program design, following the work of Fowler [17], but is also found in other disciplines, such as formal methods [18]. Refactoring carries the implication that a bad design can be recognized, and Fowler’s book has a chapter
of bad smells that serve this purpose. Refactoring includes identifying problems, selecting the target pattern, and ensuring that behaviour is preserved [19]. The refactoring described in this document is concerned with restructuring system designs, rather than programs, and the idea of bad smells is extended into Problem Templates.

It is necessary to have criteria for good or bad design in order to justify why problem templates are suspect, and why the refactorings are better. Security design principles have been understood for some time; for example, Viega and McGraw [20] list ten principles, three of which are particularly relevant:

- be reluctant to trust;
- execute all parts of the system with the least privilege possible;
- keep it simple.

To trust is to place reliance on something; any service constrained by a security control is relied on to support the security of the system, so the parts of the system that are security critical should be minimized. Privilege is the degree of access, or authority, that any process has. The ‘keep it simple’ principle is a reminder that security guarantees and functional complexity are not easily combined; complex applications should not be security critical, and interfaces to security functions should be as simple as possible.

The security principles directly suggest design tactics: ensure that protection mechanisms are as close as possible to the objects they protect, and isolate security mechanisms from other functions. This approach is also espoused by Web service architects [2]: applications should be invoked with self-contained evidence of the process history, allowing the service to judge at the point of execution if process constraints have been upheld. These measures reduce the need for enactment guarantees and bespoke access control, minimizing the number and complexity of security critical components in the overall system.

Of course, design is more complex than applying these simple tactics, and it is necessary to show how these ideas apply to representative workflow applications. This is the substance of this paper; Section 3 describes problem templates to help system designers recognize undesirable configurations, target patterns to which they can be refactored, and explanations of how these apply to workflow systems. Section 4 presents worked examples to show how these concepts are applied in practice, followed by a discussion of implementation issues.

3. REFACTORING

The refactorings in this section are concerned with the structure, or architecture, of services and data in a service-based system. They do not deal with detailed protocols or security mechanisms. Partitioning a process into services, and the data that flow between them, is a fundamental design step, equivalent in conventional projects to developing detailed operational requirements from top-level goals. It is this phase of system design to which these refactorings can be applied.

The description of each refactoring is in three parts: the problem template, the refactored solution, and a description of why the solution is relevant to workflow-oriented systems.

Problem Templates name and describe the problem, their purpose is to help non-specialists recognize system configurations that are risky from the security perspective. The term Problem Template has been used, rather than antipattern, because that term has several different interpretations in the literature.
The refactored solutions provide target patterns for good practice that are separately named and given implementation independent descriptions. The refactoring of some problems is context-dependent, so alternative target patterns are documented where necessary. The patterns assume the use of appropriate security implementation mechanisms; these are likely to include communication services, such as encryption or user authentication, protocols that can provide evidence of data authenticity, and controlled access to services.

Gamma et al. [21] established patterns as an important way to communicate design experience, and security patterns have been catalogued by the Open Group [22]. In their terms the patterns given here are structural, but since the focus of this document is workflow security, not the definition of a pattern language, the patterns are given lightweight descriptions compared to the formats that are usual in the patterns community.

Four refactorings are described below, the first two (Delegated Access, Composite Data) are concerned with confidentiality, the third (Detached Policy) with integrity, and the last (Asymmetric Evidence) with Provenance and Accountability.

Diagram conventions. Services and data items are as shown as simple stereotypes (see Figure 1). Solid arrows between services show invocation and dashed arrows indicate the movement of data. Data items are shown in their normal (i.e. persistent) positions.

In a service-based architecture all data items would be accessed via a service and this encapsulation is shown by the outline service around a data item; data managing services are shown explicitly only when they play a significant part in the description. Services and data within an outline service boundary are local to each other, but any other objects may be remote. Organization (security domain) boundaries are shown if they are significant.

3.1. Delegated access management

Delegated access management occurs when a remote service is made responsible for the distribution and access control of confidential data, and the data owner determines the access policy. The characteristic problem template is shown in Figure 2; it can be recognized by the presence of an intermediate service that manages access to confidential data items.

Of course, any service that receives a confidential data item is required to police its access, and the distribution of confidential data cannot be eliminated. However, designers should be wary of this arrangement if:

- the intermediate service has no other use for the data item; or
- the intermediate service is intentionally providing an access management service.
In either of these cases the designer should consider an alternative, since this arrangement will:

- add to the trust required in the intermediate service;
- increase the number of locations in which confidential data is stored;
- place policy synchronization requirements between the intermediate service and the original data domain.

This template has the pre-condition that the data owner determines the access policy; if not, the refactoring may not apply. For example, in the case of content filtering to remove pornography (e.g. the pattern at [23]), the administrator of a client site may wish to impose an access policy independent of the data origin; the solution in this case is to apply access controls using an intermediate service located within the administration with the restrictive access policy.

3.1.1. Refactored solution: distribution by reference

This problem can be refactored as shown in Figure 3, by distributing references rather than data items, and ensuring that they can be resolved only by the intended recipient.

This solution assumes that there is a functional need for the intermediate service to provide data to the recipient; if not, it can simply be eliminated. The data service provides a reference to the data item that is opaque, in that it neither carries content nor confers access (1); this can be safely distributed (2). The reference can then be used to request the data (3), which is provided via access controls local to the data item (4).

This allows the intermediate service to carry out a task that is functionally equivalent to data distribution, without the need to trust it with the actual data, resolving the problems in the original template.

An alternative solution to this problem is to encrypt the data; this simply transfers the problem to managing cryptographic keys rather than data items, which may be straightforward in the case of a single data recipient, or rather complex if the access policy is dynamic.
Neither this pattern, nor encryption, is able to keep the existence of data confidential; if this is a requirement, then further specialist security design is needed.

### 3.1.2. Application to workflow

Workflow sub-systems often construct lists of documents for their users; these include textual documents, database query results, and process records that range from formal contracts to informal annotations. Because a workflow sub-system serves many users, and implements many process instances, if any of this data is confidential, then the sub-system is the intermediate service in this template.

A refactored system would store references to data items (e.g. URLs) in the workflow service, and only these references would be provided to users; the ability of users to resolve the URLs then depends upon the normal system access control policy. This arrangement maintains the functional aspects of the workflow service (e.g. managing and providing work lists) while removing the need for it to be trusted to maintain the confidentiality of the data.

### 3.2. Composite data

The composite data problem arises when the granularity of data items provided by a service does not match that of the access control policy. The characteristic template, shown in Figure 4, can be recognized by the need for an intermediate service to provide filtered views (selected parts) of data items before they can be distributed.

Some types of composite data, such as databases, hold very fine grain elements; others result from the bundling of items that may be functionally related (e.g. a data item and metadata describing its provenance) but which have markedly different security concerns. This design is undesirable in both cases, although the choice of refactoring depends on the granularity.
The disadvantages of this design are similar to those for delegated access management, because this arrangement will:

- add an additional trusted service (the filter);
- increase the number of locations in which confidential data is stored;
- place policy synchronization requirements between the intermediate service and the original data domain, which may be difficult to achieve if the policy is dynamic.

3.2.1. Refactored solution: match the granularity of data to access controls

There are two ways of refactoring this problem, depending on the degree of granularity of the underlying data; both are shown in Figure 5.

The coarse grain solution (a) is preferred, and can be employed if the original data items aggregate a small number of distinct components, such as the data and metadata described above.
In this case the data should be partitioned into items that are atomic from the security as well as the functional perspective. This solution is preferable because the resulting data objects can be managed by normal system access controls, with the result that their distribution and management is straightforward.

The fine grain refactoring (b) simply incorporates the data filter in the managing service for the data item, localizing access constraints and policy management. This resolves policy synchronization and is also in keeping with integrity management for data (see Section 3.3.1) but it leaves the security designer with other problems that must be handled on a case-by-case basis—the resulting views are probably not atomic from a security perspective, so there is a risk that the underlying data could be reconstructed by inference or aggregation.

Of course, access control granularity is implementation dependent and a system designer may anticipate the availability of finer grain policy enforcement mechanisms, such as filtering of XML documents. The principle remains, however: if system stakeholders have significantly different concerns about two classes of data, then they should be separable by the fundamental access control mechanisms of the system.

3.2.2. Application to workflow

The combination of data and metadata, mentioned above, is a common feature in data-oriented systems. Metadata provides information about otherwise anonymous data sets (e.g. instrument recordings) by recording their provenance (e.g. creation date, experimental parameters, and calibration). From the integrity perspective there may be an advantage in keeping metadata and data together as a composite item, but it may also be the case that the data without its metadata is relatively meaningless. In this case, treating associated data and metadata as different items (Figure 5(a)) may result in the beneficial effect that the raw data can be processed in a distributed system with less concern about its confidentiality.

Metadata may also be also used in workflow enactment to determine the process path. It is therefore important to consider if metadata (without data) also matches this template, and should be sub-divided according to use (Figure 5(a)) or requires the fine grain pattern (Figure 5(b)). A supplementary strategy is to implement workflow decision points as external services, restricting the extent that data need to be provided to a common workflow service.

Collaboration processes often need to attach annotations to system objects, such as process results or documents, and these annotations may have independent security implications. In extreme cases they may amount to a contract or a command; in more informal circumstances they may be private comment on otherwise public data. Workflow sub-systems that handle annotations and data items as composite objects, and filter their availability depending on the user, match this problem template and should be refactored accordingly. For example, if a data item is a public document with a commercially sensitive annotation, then it should be implemented as two separate (linked) files, rather than a single compound item. This approach allows normal system access controls to police the protection requirement, rather than depending on bespoke filters to provide selective views.

After this refactoring, it may be necessary to distribute the resulting data by reference (see Section 3.1.1) to avoid the delegated access problem.

The most common example of fine composite data is a database, and localized filtering of such data (Figure 5(b)) is often included as a security feature in commercial database packages.
3.3. Detached policy

Integrity constraints limit the way data items can be changed, to preserve their content or to maintain consistency with other data, or with the external world. The detached policy problem arises when a remote service is responsible for the integrity of a data item. The characteristic template, shown in Figure 6, can be recognized by integrity requirements (data integrity policies) that are enforced by services other than those that directly manage the associated data item.

The separation of the policy enforcing service from the data item may be designed to allow:

- the managing service for the data item to service requests from many remote users; or
- the remote service to manage a range of different data types.

In the first case, the integrity of the data item depends on the policies of all the services that may access it. In the second case the remote service is relatively complex, and it may be difficult to show that the integrity policy is upheld in all possible circumstances. Either of these result in a brittle system, in that small changes to a component or policy may have unforeseen consequences for the integrity of the data.

3.3.1 Refactored solution: localize policy enforcement

Integrity management was first described by Clark and Wilson [24] and the refactoring of this problem follows their principles, which restrict data updates to well-formed transactions; this is the function of a data-managing service. However, this is not sufficient; in a distributed environment a critical factor is how close this service is to the data item. The problem in this template is that the integrity-managing service is remote, increasing the opportunity for failure.

Moving policy enforcement close to the data item it protects is the essence of the refactoring (see Figure 7(a)). In a distributed system, ‘close’ may be meaningless if the integrity policy is to maintain consistency between data items that are not co-located. This requires an elaboration of the preferred pattern, shown in Figure 7(b).
In the preferred solution (Figure 7(a)), the only means of accessing a data item is via a local service that enforces the whole integrity policy for that item. The benefits of using a local service to ensure integrity are:

- the minimum number of components are needed to implement the integrity policy;
- the policy is expressed in a single place;
- the policy is enforced in the same administrative environment as the owner or custodian of the data.

An important variation of this pattern is shown in Figure 7(b). When the policy requires consistency between data items that are not co-located, then the best solution is to factor the policy into rules about specific data items, and the need for agreement across the distributed system. The former can be located close to the managed data items using the preferred pattern; the distributed constraint can be reduced to requiring agreement between two or more managing services before updating their respective data items.

The benefit of this arrangement is that the distributed function can be simple and uniform. This allows the Agreement Service in Figure 7(b) to be implemented as infrastructure, rather than an application specific mechanism, and the primary integrity controls to remain local to the data items they protect.

### 3.3.2. Application to workflow

The implementation of process constraints (see Section 2) needs to be considered in the light of this template. They often amount to data integrity requirements, but if they are enforced in the workflow component by constraining the plan or its enactment, rather than by services local to the data, then the design matches this problem template.

For example, consider the requirement for separation of duties (e.g. different Approval and Payment officers in a payment approval scenario). This could be enforced at the payment service with an
access restriction that the service invoker and approver are different. The process of finding suitable
people to complete the tasks of approval and payment may be a complex planning problem, but policy
enforcement can be disassociated from this complexity and moved close to the service that makes the
payment.

Refactoring in this way requires the workflow system to provide evidence of the approval and
payment authorizations to the payment service, and this exposes the need for integrity in process
records. The straightforward solution is to regard critical process records (e.g. those used for
authorization or evidence) themselves as documents, and therefore subject to this pattern. There are
further benefits in recording critical decisions as documents, these are discussed in Section 3.4.2 and
featured in the worked examples in Section 4.

The most widespread example of an Agreement Service (Figure 7(b)) is the two-phase commit
protocol used between database systems to ensure that the data have been updated consistently, or not
at all. In this case the Agreement function is synchronous, but there are also examples of asynchronous
Agreement functions (e.g. BPEL4WS [3]); generally, one of the updates is allowed to proceed, and
if the second fails then the managing service for the first data item is expected to be able to undo its
update.

3.4. Asymmetric record

The asymmetric record problem arises when two parties (organizations, projects, individuals) with
differing interests collaborate to produce data, but where the record is maintained by just one of them.
The characteristic template is shown in Figure 8.

This template is significant if:

- the two parties are in different administrative domains; and
- the evidence is a matter of record for them both.

Data is a matter of record if it is valuable to both parties; for example, if it records a contract or its
execution. This encompasses a broad variety of records including recording, accounting, and charging
for the use of equipment or specialized services, and access to electronic products such as journals.
Not all data of these types matter: organizations do not choose to account for all the services they
provide and may regard some accounts as of marginal value. However, in circumstances where they
may be a dispute between two organizations about their interaction, systems designed in this way deny critical data to one of the parties.

This problem is intuitively obvious, but it is sometimes difficult to identify. The reason is that it can be recognized only in the light of a security asset risk analysis that determines what assets matter to which parties and the respective threats and impacts of loss to each. Security risk analysis is not always integrated in distributed system engineering, resulting in this type of security anomaly.

3.4.1. Refactored solutions: distributed evidence, or a trusted third party

There are two ways that this problem can be refactored; both are shown in Figure 9. These solutions are described using two parties, but they can readily be extended to multiple domains.

The preferred refactoring is for critical data flowing between the domains to be self-documenting to such an extent that both parties are left with an adequate record. The two evidence records do not have to be identical, although often they will be. This philosophy is consistent with Web services architectures, where it is expected that the invoker of a service provides all necessary evidence of entitlement \[2\]. If this general principle is followed, then recording evidence of the interaction can follow naturally.

The second option (Figure 9(b)) may be needed to meet unusual security requirements, such as mutual anonymity or unlinkability between the participating domains. For example, trustees feature in some electronic cash systems \[25\]. The use of a trusted third party does not, in general, reduce the quality of the evidence that service providers need to supply, so this should only be used if it is indicated by specific requirements for third parties, for example to implement escrows.

3.4.2. Application to workflow

Workflow systems may manage formal contractual flows (e.g. purchase orders), or collaborations that involve commands, instructions, or decisions being passed between collaborating companies. For example, DAME \[1\] passes annotations on maintenance cases that range from informal comments to explicit recommendations. Annotations of this sort may be the subject of later dispute, perhaps to
demonstrate contract compliance or due diligence. Even records of resource usage or tasking may need subsequent validation if one of the parties suspects misuse of their resources.

In these circumstances it is obvious that the record of interaction between the parties cannot be held by any single participant (Figure 8) and that the system should be designed to provide each with a useable record of any critical interactions (Figure 9(a)).

The Detached Policy template (see Section 3.3), introduced the need for documents with self-contained authentication to ensure the integrity of process records that authorize action; the need for an authoritative record of the interaction between two parties is further motivation for this method of process recording. Web services standards allow documents to be incrementally written and signed [26]. This can be used to document a process in a manner that can be transported, stored, referenced, and indexed. It also allows compatibility with policy enforcement points that can extract this information to enforce security decisions.

The preservation of a workflow record as a series of documents, rather than objects within the enactment system, provides further benefits, since it:

- ensures that normal protection regimes apply to access control;
- provides a natural provenance recording mechanism;
- provides a standard mechanism for transporting ‘work in progress’ across organizational and security boundaries;
- is intuitively similar to existing business processes that are enacted by sequences of signed documents [27].

These concerns apply to workflows that span domains, but the same principles may be applied to service invocation within a domain. The former is a question of contract or process provenance, the latter may be useful to provide a chain of accountability for audit purposes.

3.5. Summary

This section has provided problem templates that allow developers to recognize functional configurations that are poor practice from a security perspective, and patterns that show how they should be refactored. For workflow systems, the resulting design guidance can be summarized as follows.

- Pass data by reference, not value. Where possible, document access should be managed via normal distributed access controls.
- Match the granularity of documents to the access control system. If necessary partition composite documents (such as data and metadata) to allow access to be managed separately.
- Move integrity management close to the data it protects. Data integrity should be managed by a local managing service, not by constraints on a remote enactment process.
- Remote application invocation must be able to demonstrate authorization. Where processes span multiple security authorities or organizations, each invocation across the organization boundary must be self-documenting, and allow the remote domain to establish the basis of accountability on which it is to act.
Applying this guidance systematically separates the functionality of a workflow system from the need to trust it to provide security guarantees. The next section shows how this guidance can be used in practice.

4. PRACTICAL EXAMPLES OF DISTRIBUTED WORKFLOWS

This section provides two examples to show how the patterns described above are used. The first is a payment system, which includes a formal chain of authorization and a separation of duty constraint, the second is a collaborative workflow between two organizations, where the exchange is a matter of record for both. The section closes with a brief discussion of implementation issues, in particular when these patterns can be applied and when it is difficult.

4.1. Payment process

This example describes an authorization and payment system, including separation of duty, which is a significant feature in workflow security. The scenario is: any company employee can raise a payment order, but it must then be authorized and the payment approved before the accounting system makes the payment. In order to reduce the risk of fraud, different people must carry out the authorization and approval steps, although managers in the company have the ability to do either. The status and details of the payment order should be visible to the initiator and to company managers only.

The role of the workflow service is to ensure that the process is carried out in a timely and efficient manner. This example describes normal operation, but a real process would be more complex: it would ensure timeliness by selecting users who are available, allow escalation of difficult cases to a manager, and handle rejected requests.

A document of record provides a transportable and secure account of the process, while still allowing it to be managed by a workflow service. The process produces and delivers payment requests, so these documents stand as process records. The documents can be stored in a standard access protected file system, and digital signatures can be used to signify authorizations that are eventually consumed by the accounting system. The overall process is shown in Figure 10.

The process is started by a user who creates a payment request (1) and invokes the workflow service by providing its reference (2). The workflow then provides the reference to a user who is permitted to authorize payments (3a) and who has the authority to access the document and digitally sign it (3b); the same procedure is carried out for approval (4a, 4b). When these tasks have been completed, the workflow is able to present the reference to the accounting system for action (5). The accounting system is able to obtain the document (6) and check that it has been authorized and approved before making the payment; finally the accounting system appends a record to the payment request indicating that the task is complete (7).

The security protection in this system is located in the access control to the document file, the signing process, and the accounting service.

The document can be created by any user, who restricts its access to the originator and to company managers. At this stage there is no integrity requirement, any of those with access can change the document prior to its authorization.
When a manager wishes to authorize or approve a payment request, the document is retrieved, digitally signed, and replaced. In this way the approver countersigns the authorization signature as well as the original document. This process provides payment authorization and also ensures the subsequent integrity of the document: if the document is changed after it has been signed, then this can be detected. The originator does not sign the request, because there is no security requirement to do so; evidence is only recorded when it is needed, not for every action.

The invocation of the accounting system by the workflow service does not authorize the action; the accounting service makes the payment and carries out the associated bookkeeping if it is satisfied that two different managers have signed the payment request. Writing a completion annotation to the request document allows the accounting service to confirm completion to the originator, without providing access to company ledgers.

From the security perspective this design separates the workflow functionality from data and process protection mechanisms; it also naturally provides a transparent incremental journal of payment requests and fully protects the company accounting system.

This design uses three of the four patterns described in the previous section: documents are managed by reference, process integrity (including separation of duties) is managed close to the system that makes the payment, and the invocation of the payment service references a request that is self-contained with the authority and accountability needed to permit the action.

4.2. Formal collaboration

In this example, a formal process flows between two workflow systems situated in different organizations. The interaction is contractually significant in some way; it requires one or more of the partners to base its decisions on propriety data, which must be protected. Interactions of this sort are common in information-based Grid applications; for example, collaborators use the DAME [1] system to exchange potentially costly advice about maintenance actions required on aero-engines, based on propriety databases that record engine performance and diagnostic histories.
A worker in the first domain is alerted to the output of an automated process, for example a data query, and needs to request action from a co-worker in a different domain. This collaboration is formal, in the sense that both parties may need evidence of the interaction to show contractual compliance, but the query result that triggers the exchange is confidential to the first domain. The process continues with other workflow actions and perhaps the return of the process thread to the original domain, but this fragment is sufficient to show how to construct a formal interaction between two domains.

The role of the workflow service is to automate the processing that produces the query result, alert the user, and then act on the user's decision to transfer processing to a different organization. In this case the document of record is an annotation; the overall process is shown in Figure 11.

The workflow invokes a Query service on some local data (1), which creates and stores a result set (2). The service returns a result reference to the workflow, perhaps with limited metadata to allow the workflow to determine the next action (3). The workflow determines the need for a user decision, so a user is alerted (4) and provided with the result reference. The user reads the result data (5), determines a course of action that is then recorded in a signed annotation (6), and provides its reference to the workflow (7), together with routing information. The workflow routes the reference of the annotation to the workflow invocation service in the second domain (8), which is able to retrieve the annotation (9), check and record the authorization, and invoke its local workflow (10).

The security protection mechanisms are in the access control to the document file, the signing process, and the workflow invocation service in the second domain. The design of the process ensures that the annotation is separate from the result data, allowing the document access control to limit remote access to information that can be shared. Standard protection mechanisms (firewalls, access control) are used to prevent direct external access to any other system components.

The workflow service in domain B is protected by an invocation service that is able to confirm proof of authorization in the form of a remote document, before continuing the process thread. The purpose of the authorization document is to provide proof that the request originated from an authorized individual in the first domain. As with all cross-domain interactions there is a prior trust arrangement that allows domain B to check the authenticity of the authorization; for example, certificates issued at domain A may be traceable to a Certificate Authority that is acceptable to domain B, and domain A may publish
details of the individuals authorized to sign the request in a publicly-accessible directory. (For a more
detailed description of a possible workflow trust arrangement, see Milosevic et al. [27].)

An important feature of this architecture is that both domains have evidence of the interaction, and of
the authority by which it was invoked.

This design uses all the patterns described in the last section: the workflow systems manage data
by reference and are therefore not required to protect query results; the annotation is kept separate
from the result, thus controlling the granularity of data objects to ensure that they can be managed
by the document access control system; integrity protection is close to the component it protects, in
both the query service and the second domain; finally, the remote invocation is self-documenting.
This demonstrates that these patterns are able to separate security and workflow functionality even
when the process spans different organizations.

4.3. Implementation issues

This paper has shown that it is possible to recognize good and bad security design, and that good design
practice does not just work for isolated patterns, but can be applied to complete workflow applications.
However, in practice there may be a need to compromise between security and other concerns, and this
section describes some common issues:

- trust in the information required for workflow routing;
- performance of the workflow system;
- performance of the security infrastructure.

Routing information. Section 3.2.2 noted the importance of information used by the workflow service to
make routing decisions; the problem is that the presence of this information may require the workflow
system to be trusted.

The strategies in this paper can always be used if the trust requirement is integrity: security is about
essential guarantees not correct performance, and the two can be separated. Integrity guarantees can be
implemented outside the enactment system, even if the same information is required inside the system
to ensure correct routing. The two examples above illustrate this process.

Confidentiality is more difficult; if an item of data is essential for routing, and it is confidential,
then the workflow system must protect that confidentiality. In some systems it may be possible to
move routing decisions to an external component or service, but the more general strategy is simply
to minimize the trust requirement to that which is essential for enactment. For example, just because
the workflow system needs information for routing decisions, there is no need for it to manage the
availability of that information to users—that can still be done outside the workflow service. Trust is
not a binary measure: it can still be minimized even if there are residual requirements.

Performance of the workflow system. The core concern is usually the rate at which routing
decisions can be made, and often this depends upon the speed with which the routing data can be
accessed. The patterns proposed here are either neutral, or improve the core enactment performance.
Minimizing the data handled by the enactment system reduces the size of the supporting database, often
increasing the proportion of routing information that can be memory-cached. Removing other forms
of trust, such as user data management, reduces the security overheads in the enactment component.
For example, passing data by reference (e.g. URLs in place of whole pages, see Section 3.1.2) is common for performance reasons and version management, as well as security.

Performance of the security infrastructure. Removing security from the enactment system tends to improve its performance, but it does introduce overheads elsewhere in the system. In some cases there will be little extra cost, since the components will already support the necessary security functions. For example, in the data-by-reference case, the original data repository will provide the same security, and similar volume handling, regardless of the routing of the data. However, the integrity patterns require cryptographic signatures, and these operations are computationally expensive. Most signature operations will be invoked by users, in which case their latency is probably acceptable (i.e. the latency of the security operation is insignificant when compared with the latency of invoking the user). If this is not the case, then designers need to carefully balance the design of the security system against the need for system throughput.

In summary, there are two potentially difficult implementation cases: when the workflow routing data is itself confidential, and if the overhead of integrity signatures is significant in system terms. In most cases, however, the design approaches proposed in this paper can be applied easily, and may even enhance system performance.

5. CONCLUSIONS

This paper has shown that workflow security requirements can be supported by standard distributed-system protection mechanisms. This is preferable to integrating security functionality in workflow enactment components, because it reduces the need to trust complex workflow software, uses protection mechanisms that are integrated with the security policies of their domains, and allows workflow components to be deployed as services, rather than privileged system components.

The paper documents four design issues, or problem templates, that illustrate common system problems; each is refactored into a pattern of good practice. The documentation of these patterns is focused on workflow systems, but they also have wider application in service-based architectures. The paper concludes with two worked examples that show how these patterns can be combined to implement realistic processes: a payment workflow that demonstrates separation of duty; and a formal collaboration that shows a process shared between different organizations. Separation of duty is a common workflow security issue, and the collaborative process is typical of information-based Grid systems.

These examples focus on formal business processes, because they provide straightforward examples of protection requirements; however, the principles apply equally to informal collaborative arrangements, assuming that such arrangements have security concerns. Security controls (e.g. integrity signatures) are only used to support the protection requirements of the system, not to document every process step.

Implementation issues are summarised in Section 4.3; two potential conflicts between security and functionality are noted, but in general the application of these principles is straightforward, and may even improve system performance. The main practical issue is the need for system designers to consider security early in the design process, and this is already essential for most distributed or Grid-based systems. Key design issues are matching the granularity of data to security concerns, the distribution
of references in place of data items, and recording the authority for actions in self-contained process records.

ACKNOWLEDGEMENTS

This work was first presented at the Global Grid Forum 10 Workflow Workshop; it benefited from valuable questions from many attendees at that event, and from constructive comments and reviews from Susan Stepney and from the anonymous referees. This work was supported by the Royal Academy of Engineering, and the security study that prompted its development was carried out in conjunction with the DAME team, including staff at Rolls-Royce, Data Systems & Solutions, Cybula Ltd, and the Universities of York, Leeds, Sheffield, and Oxford.

REFERENCES


