Security Mechanisms for Workflows in Service-Oriented Architectures

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Jens Müller

aus Henstedt-Ulzburg

Tag der mündlichen Prüfung:	16. Januar 2015
Erster Gutachter:	Prof. Dr.-Ing. Klemens Böhm
Zweiter Gutachter:	Prof. David W. Chadwick, PhD

Acknowledgement

The completion of this thesis would not have been possible without the support of and exchange with so many people to whom I owe lots of gratitude.

First of all, I would like to thank my advisor Prof. Klemens Böhm. He guided me with patience and encouragement. His comments on the drafts of my publications were plenty and always helpful. Thanks also go to my co-advisor, Prof. David Chadwick for his helpful suggestions regarding security mechanisms, especially security architectures.

I also thank my colleagues at the *Information Systems Group* for fruitful discussions and the good working atmosphere. I only mention the TAS^3 project team here: Thorsten Haberecht, Jutta Mülle, Dr. Silvia von Stackelberg and my office mate Dr. Christian Hütter. I also want to thank my other colleagues from the TAS^3 project. The many occasions when we met are certainly ones I will remember.

My special thanks go to my parents and my friends Dorothée, Hanka, Henning, Katrin, Oliver and Susanne, who always had an open ear for me and helped me with their advice or just by listening.

Kurzfassung

Motivation und Zielsetzung

Zur Beherrschung der immer komplexeren Informationverarbeitung und gleichzeitiger Bewahrung von Flexibilität sind neue Paradigmen notwendig. Für die dabei gerade in verteilten Umgebungen auftretenden Probleme sind *serviceorientierte Architekturen* ein vielversprechender Ansatz. Diese basieren auf *Services* (Diensten) mit abgrenzbarer und gekapselter Funktionalität, die sich unabhängig von anderen Diensten entwickeln können. Service-Kompositionen auf hoher Abstraktionsebene zur Umsetzung von Geschäftsprozessen einer Organisation werden durch Workflow-Technologie ermöglicht.

Insbesondere in Anwendungen mit sensiblen personenbezogenen Daten ist Sicherheit wichtig. Diese muss in Workflow-Management-Systeme integriert werden, um sichere Service-Kompositionen zu gewährleisten. Um benutzerzentrierte Workflows möglich zu machen, ist auch die Integration identitätsbasierter Services erforderlich. Damit dieses Ziel erreicht werden kann, ist zunächst eine systematische Analyse der entsprechenden Sicherheitsanforderungen notwendig. Darauf kann dann der Grobentwurf einer entsprechenden Architektur aufsetzen. In technischer Hinsicht ist außerdem das Zusammenspiel mit existierenden Spezifikationen und deren Implementierungen zu klären. Dies ermöglicht Sicherheit in einer größeren serviceorientierten Gesamtarchitektur.

Insgesamt ist die in der vorliegenden Arbeit betrachtete Forschungsfrage, wie sich Unterstützung für Sicherheit und Identitätsmanagement in ein Workflow-Management-System integrieren lässt. Dabei stehen Anwendungen im Zusammenhang mit personenbezogenen Daten im Fokus. Teilprobleme stellen die Ermittlung der Sicherheitsanforderungen von Workflow-Anwendungen und die Bewertung existierender Ansätze, der Entwurf einer entsprechenden Architektur, sowie die Untersuchung von Verbesserung für die Privatheit von Benutzern in der Praxis dar.

Herausforderungen

Auf unterschiedlichen Ebenen ergeben sich eine Reihe von Herausforderungen. Dies erfordert eine sorgfältige Kombination verschiedener Methoden.

Die Anforderungsanalyse muss zum einen auf einer realen Anwendung basieren, um realistische Anforderungen zu liefern. Zugleich soll das Ergebnis repräsentativ und generisch anwendbar sein. Diese widersprüchlichen Ziele erfordern einen kombinierten Ansatz basierend auf einer sorgfältig ausgesuchten Anwendung und einer ergänzenden strukturierten Analyse auf Basis von abschließend aufzählbaren Kriterien. Bei der Bewertung hinsichtlich des Stands

der Technik stellt die große Zahl von Lösungsvorschlägen in diesem Gebiet, deren Unterschiede in Abstraktionsniveau und Ausgereiftheit sowie ihre Ausrichtung auf jeweils unterschiedliche Facetten der in dieser Arbeit betrachteten Probleme sowie zum Teil auf andere Einsatzfelder eine besondere Herausforderung dar. Entsprechendes gilt auch für die Entwicklung der Architektur. Diese muss bekannte und neue Lösungen integrieren, Entscheidungen von Benutzern (auch in Form von Policies) einbeziehen und sich durch Nutzung von Kontextinformationen flexibel an die jeweilige Situation anpassen. Insbesondere ist es ggf. nötig, die anzuwendenden Sicherheitsspezifikationen dynamisch zu bestimmen. Außerdem muss ihre Struktur generisch, also nicht auf eine bestimmte Technologie festgelegt, sein. Nicht zuletzt gibt es eine Lücke zwischen einer notwendigerweise abstrakt und generisch gehaltenen Architektur und darauf basierenden konkreten Anwendungen, die einen Zusatznutzen für die Privatheit und Sicherheit von Endanwendern bieten können. Die Herausforderung besteht darin zu zeigen, dass die Architektur solche Anwendungen ermöglicht.

Beiträge der Arbeit

Der erste Beitrag der vorliegenden Arbeit besteht in einer Anforderungsanalyse anhand eines Beispiels aus der beruflichen Weiterbildung. Ergänzend werden die Einteilung von Sicherheitseigenschaften aus [ITU95c] sowie wohlbekannte Workflow-Aspekte (vgl. [JB96]) herangezogen. Den zweiten Beitrag stellt der Abgleich der Anforderungen mit dem Stand der Technik dar. Ein Schwerpunkt liegt dabei auf Zugriffskontrollmechanismen mit Bezug auf verschiedene Workflow-Aspekte. Es zeigt sich eine nicht ausreichende Berücksichtigung von Workflow-Kontext in existierenden Ansätzen für Sicherheit in serviceorientierten Architekturen.

Der dritte und wichtigste Beitrag der Arbeit ist die Entwicklung der Architektur. Diese erfolgt in zwei Schritten. Im ersten Schritt geht es um die sichere Ausführung von Workflows in allgemeinen. Um bestehende Workflow-Engines nutzen zu können, wird ein aspektorientierter Ansatz mit entsprechenden Transformationen der Workflow-Definitionen beschrieben. In einem zweiten Schritt geht es um die Integration mit Identitätsmanagement-Systemen. Für die einzelnen Konzepte werden dafür der Fluss von Identitätsinformationen durch das Workflow-Management-System sowie der Einfluss von Workflow-Kontext untersucht. In beiden Teilen werden Entwurfsgrundsätze aufgestellt und verschiedene Entwurfsalternativen untersucht. Außerdem werden Konfigurationsmechanismen entwickelt.

Insgesamt werden damit neue Anwendungen mit verbesserter Sicherheit und Privatsphäre ermöglicht. Beispielhaft wurde der Prototyp eines Werkzeugs implementiert und evaluiert, mit dem Benutzer die Workflow-Ausführung nachvollziehen können. Das Thema „Sicheres Workflow-Management in serviceorientierten Architekturen“ wurde damit umfassend von der Anforderungsanalyse bis hin zu Anwendungen behandelt.

Aufbau

Nach der Einleitung in Kapitel 1 werden in Kapitel 2 die Grundlagen erläutert. Dazu gehören die Charakteristika von serviceorientierten Architekturen sowie

ihre technische Umsetzung mit Web-Services und der Einsatz von Workflow-Management für Service-Kompositionen, außerdem ausführbare Workflows und Workflow-Aspekte. Was Sicherheitsaspekte angeht, werden verschiedene Definitionen und Einteilungen vorgestellt. Besonders wird auf Zugriffskontrolle und Identitätsmanagement eingegangen. Kapitel 3 stellt verwandte Arbeiten vor, u.a. für Zugriffskontrolle in dynamischen Umgebungen und Workflows (insbesondere Delegation) sowie Sicherheitsarchitekturen für Workflows.

Kapitel 4 stellt Anwendungsfälle vor, insbesondere aus dem Bereich Employability. Die Anforderungsanalyse in Kapitel 5 ergibt 44 generelle Sicherheitsanforderungen. Ein Schwerpunkt liegt auf der Verwendung von Workflow-Zustand für Sicherheitsentscheidungen sowie organisationsübergreifenden Umgebungen. Die Analyse des Stands der Technik ergibt eine mangelnde Integration mit Workflows. Hinsichtlich föderativen Identitätsmanagements (FIM) wird die Integration mit FIM-Konzepten u.a. anhand von Datenfluss und nötigem Kontext untersucht.

In Kapitel 6 wird die Architektur entworfen (vgl. Schaubild 6.2). Für eine Auswahl der Anforderungen, die vielfältige Sicherheitsfunktionalität abdeckt, wird die Integration in die bestehende Architektur anhand der Auswirkungen verschiedener Alternativen auf Aufbau und Schnittstellen betrachtet. Im zweiten Schritt wird Unterstützung für föderatives Identitätsmanagement hinzugefügt. Die technologiespezifischen Aspekte werden in einer Bibliothek gekapselt. Identitätsinformationen werden an den Schnittstellen nach außen durch zwei *Policy Enforcement Points* erfasst. Diese setzen auch die Sicherheitsentscheidungen eines gemeinsamen *Policy Decision Point* durch. Die Erfassung und Nutzung für Anwendungszwecke und Zugriffskontrolle werden beschrieben. Die Erläuterung der technischen Aspekte der Implementierung ist Gegenstand von Kapitel 7. Insbesondere werden Datenformate und Schemata für eindeutige Referenzen auf Daten definiert. Die Interaktion der Komponenten wird analysiert, und es werden entsprechende Nachrichtenformate entwickelt. Außerdem werden die nötigen Transformationen der Workflow-Definitionen beschrieben, und es wird darauf eingegangen, wie Performance-Engpässe behoben werden können.

Die Evaluation in Kapitel 8 weist nach, dass die Architektur die Erfüllung der Sicherheitsanforderungen weniger kompliziert und fehlerträchtig macht. Außerdem zeigen Evolutionsszenarien ihre Flexibilität. Der Prototyp einer Anwendung zur Nachverfolgung von Workflows, der in einer Benutzerstudie getestet wurde, demonstriert außerdem den praktischen Nutzen der Architektur. Kapitel 9 wirft schließlich einen Ausblick auf zukünftige Arbeiten in den Bereichen Standardisierung, Verfeinerung von Konzepten sowie Benutzerschnittstellen.

Contents

Detailed List of Contents

Chapter 1

Introduction

1.1 Motivation and Goal

Information processing systems have become increasingly complex and connected, and continue to become even more so. As a consequence, new paradigms were necessary to manage this complexity while preserving the necessary flexibility. This concerns, for example, data formats, distribution of data and processing of data. Think of a *curriculum vitae* and the information connected to it, such as university diplomas and letters of reference from past employers. The grades in a diploma are harmonised by frameworks such as the European Credit Transfer and Accumulation System (ECTS). The information comes from different authoritative sources such as universities and employers. We can imagine a scenario where the contents of a *curriculum vitae* are stored in a central repository (known as an e-portfolio) on behalf of the individual concerned. In this scenario, it would be possible for authoritative sources of information to add this information to the e-portfolio directly. It would also be possible to verify the authenticity of the e-portfolio's contents automatically and provide aggregated views of the content. Service-oriented architectures are a promising approach towards this problem. They are based on the notion of services – autonomous pieces of functionality which are linked through well-defined interfaces but can still evolve independently. In addition, messages between services are treated as first-class entities, allowing the implementation of powerful middleware functionality. For example, the middleware could automatically locate the correct repository for storing a specific individual's *curriculum vitae* information. While the independence of services is an important issue, organisations must be able to coordinate these independent services in a way that allows them to achieve their business goals and carry out their business processes. Workflow technology is able to achieve high-level service compositions by coordinating the data and control flow between services and human activities.

Security is an important issue in service-oriented architectures, in particular for applications that handle sensitive personal information. It is necessary to integrate security functionality into workflows in order to achieve secure service compositions. Moreover, workflows integrate the activities of both automated services and human participants. The participation of humans requires adaptation to their security requirements. To this end, workflows have to take

the identities of participants into account, and integrate with other identity-based services in order to achieve user-centric service compositions. Think of a service that recommends career development options to an individual. It first needs to collect information about this individual from various sources. Evaluating this information may also involve other services which must be acceptable both to the individual concerned, e. g. in relation to privacy, and to the recommendation service with respect to the reliability of the results and other quality-of-service parameters. Finally, the service stores, inter alia, recommended vocational education courses in a central repository. This certificate then serves as a prerequisite for funding by the employment administration or by the individual's employer as provided for in a collective labour agreement. To this end, it must be possible to establish the authenticity of the certificate.

To implement a system like this, a systematic analysis of the security requirements of a workflow management system in such a setting is needed. Such an analysis is an important prerequisite to make advanced security concepts for workflows explicit and enforce them consistently.

Based on the resulting understanding of the security requirements in question, the next step is to determine what the architecture of a workflow management system that can meet these requirements must look like. This is a task we can approach at different levels of abstraction. From a high-level perspective, the components of the system, their interaction and the interfaces of the system have to be specified. For a general purpose system, configuration mechanisms are needed as well. On a more concrete level, one has to specify how the system implements relevant existing specifications, especially how it interacts with other implementations of these specifications. This is because the scope of the system designed is necessarily limited, and it can achieve security only in connection with other systems.

Using workflows in such settings, i. e. to orchestrate user-centric service interactions, enables new opportunities for users to understand and influence the workflows they participate in. For example, it becomes easier to present users with a live view of workflows processing information about them. Such applications use the infrastructure provided by a security- and identity-management-enabled workflow management system.

Overall, the research question addressed in this thesis is how to integrate support for security and identity management into a workflow management system, with a focus on applications dealing with personal information. This question can be broken down into three parts as follows:

- What are the requirements of workflow applications of this kind with respect to security? Which solutions to these requirements already exist?

- What does the architecture of a workflow management system resulting from these requirements look like? What is the general structure of the system, i. e. which components are there and how do they interact? How can it be embedded into environments based on existing standards?

- How can such a system facilitate user privacy through increased understanding and awareness, and more ways to control the execution of workflows?

1.2 Challenges

This research question and its various facets pose a number of challenges, as explained in the following. These challenges are of different natures and on different levels, and had to be approached by carefully combining different methods.

First, the analysis of security requirements has to be based on real-world applications to find requirements that are relevant in practice. However, the analysis should also yield requirements representative for a generic class of applications. This means we have to pursue two aims that may seem contradictory at first glance. Nevertheless, it is possible to achieve both goals equally using a combined approach: First, one has to use a carefully selected real-world application that is representative for the generic class of applications dealing with personal data. Second, one has to perform a structured analysis based on enumerable criteria, as this makes it possible to consider requirements that are not obvious in the chosen application, but have to be accounted for in order to yield a system with comprehensive functionality.

Second, another set of challenges arises from the assessment of the requirements with respect to the state of the art. It is not clear which state of the art should be considered. There are proposals in academic literature, and also industrial proposals. Existing approaches are specified on varying levels of abstraction and to varying degrees of completeness, and they address different facets of the problems involved. In addition, they often were intended for different scenarios than the ones investigated here, so their applicability is not obvious.

Third, coming up with an architecture that integrates existing and new solutions for the security requirements that have been pinpointed is also difficult. Workflows dealing with personal data in some cases require involving the user at runtime. Think of domains like e-health and e-employability where sensitive personal information accrues. The consent of affected users (data subjects) to the processing of their data is desirable or even required by law. This and other forms of security-relevant information provided by users during execution of a workflow is important for security enforcement. However, it is also tedious for users to frequently express consent and make choices manually, so that automatic, policy-based solutions are needed. Another challenge arises from the open nature of SOA, which makes it impossible to anticipate every situation and define how the WfMS should behave to ensure security. The WfMS rather has to adapt, by taking context into account. Such context is only available at runtime. For instance, the history of activities is necessary to enforce constraints such as *separation of duty* (SoD). As application services are discovered at runtime, it is important that a trust relationship with a service is established in a secure way. Different kinds of information, such as feedback on the past behaviour of a service, can play a role [BEdH+10]. It is thus mandatory to use the runtime context of a workflow, combined with external information and explicit decisions of users, to make security decisions. Such context can include users and application services involved, their properties, and the data and control flow of the workflow. Sometimes even security specifications themselves depend on the context, such as a policy depending on the specific user whose data is transmitted to an external service. Accordingly, the architecture has to deal with user feedback and contextual information, i. e. capture, distribute,

store and use it, and the resulting architectural requirements and effects are not obvious. Another challenge is that the architecture should be generic and that it should abstract from specific protocols. Failure to achieve this would inhibit the integration of existing security technology. Any ad-hoc modification of a workflow management system to accomodate a specific technology also carries the risk of being specific to this technology only. Instead, our goal is an architecture that can accomodate new technologies for a certain security goal while maintaining its general structure.

Additional challenges with respect to the architecture arise from the integration of federated identity management (FIM) technologies. To ensure user privacy, the resulting system must support the respective FIM features, like identity mapping between pseudonyms, prevent leakage of identity information between workflow instances, and provide appropriate user control over the disclosure of personal data. As with other security mechanisms, the implementation of FIM features needs to take the workflow context into account and adjust to the requirements of individual workflow definitions.

Finally, there is a gap between the architecture, which is necessarily rather abstract and generic, and concrete applications with added value for end-user privacy and security. The challenge is to show that the architecture directly enables such applications.

1.3 Contributions to the Field

Our first contribution is a requirements analysis (Subsection 5.1.2). We study a real-life business process, the *Accreditation of Prior Learning* process (Section 4.1). Two factors drive the identification of gaps in the requirements: Regarding existing security mechanisms in SOAs, we rely on security facets, defined in ITU-T Recommendation X.810 [ITU95c]. With respect to the workflow management domain, we consider well-known workflow aspects (e.g. the functional, informational, behavioural, procedural and organisational aspects, see [JB96], [CM06]).

Secondly, we relate these requirements to the state of the art in Subsection 5.1.4 and identify gaps not yet addressed in literature, as well as existing solutions. We group existing approaches according to different security mechanisms, such as access control, authentication and auditing. One focus is on access control. We distinguish groups of access control mechanisms for different workflow aspects. The result of the analysis is that current security solutions for SOAs are not sufficient because they do not solve our requirements appropriately, as we will explain in detail. Mechanisms such as WS-Security [LK06b] consider messaging between web services, but not the workflow-specific aspects. This is insufficient, as the entire control flow of a workflow should be subject to security constraints. Existing workflow management systems (WfMS) and the corresponding standards are quite mature with respect to the core functionality, but they provide only basic security functionality. In particular, access control is only supported on the level of single web-service calls, which is not suitable for use cases with actors and multiple web services involved. Further, they do not take the workflow context (e.g. active sub-workflow, responsible user) into account when making access control decisions.

Thirdly, we develop the architecture of a WfMS with support for privacy

and security, in two steps. The first step concerns the anatomy of a WfMS that allows the secure execution of workflows. In detail: We develop design criteria a secure WfMS should follow. For instance, it should respect established reference models and be able to incorporate legacy systems. We identify design alternatives in relation to our overall goal. We say which new components are necessary to provide a secure WfMS, and how components interact. For the use of legacy workflow execution engines, we advocate an aspect-oriented approach. More precisely, we define how to transform workflow definitions before they are deployed to a workflow engine to integrate them with the components of our architecture. We outline how to derive the security configuration of the WfMS from security annotations to workflow models. The second step then address the integration of identity management in more detail. In particular, we determine how concepts of federated identity management (FIM) can be used in workflows. We do so by analysing the flow of identity information and the influence of workflow context on FIM functionality. We also describe peculiarities that arise when combining FIM concepts with workflow management concepts. As in the first step, we consider different design alternatives and develop configuration mechanisms.

All in all, our work paves the way for the systematic implementation of a secure workflow management system and enables new applications that enhance security and privacy.

We have designed one such application, namely, an auditing tool dubbed WoSec (*Wo*rkflow *Sec*urity). We have implemented a prototype of this application and have evaluated it with real users. More specifically, our contributions are as follows: We have analysed which information must be provided to users in order to audit data transfer in distributed applications, and how it can be presented visually. We have implemented a web-based tool for auditing the handling of personal data that works with graphical representations of BPMN models of applications. It allows users to "follow their data" when it is transferred to another organisation that also provides data to WoSec. It can also be used to visualise how an organisation *intends* to handle personal data, enabling users to give more informed consent. We propose an extension to a secure workflow management system that complements this auditing tool by providing it with information on relevant events. We have designed several example use cases for distributed data processing that are sufficiently complex for a realistic evaluation: applying for an internship, trading items on an online marketplace, and buying a car. In a user study we have designed, we evaluated the tool and various features of it. The study showed that that users prefer graphical audit facilities, and that these lead to a better understanding of data transfers.

To summarise, we have comprehensively addressed the topic of secure workflow management in service-oriented architectures from a variety of angles – from requirements engineering and assessment of the state of the art, through to architecture on different levels and applications of the resulting architecture.

1.4 Outline

The rest of this thesis is structured as follows:

Chapter 2 introduces fundamentals from all relevant areas, and Chapter 3 discusses related work. Chapter 4 introduces the use cases.

Chapter 5 features a requirements analysis and an assessment of the requirements found with respect to the state of the art. Chapter 6 presents the system design and architecture, while Chapter 7 addresses the implementation in more detail. Chapter 8 comprises an evaluation using a range of criteria. Finally, Chapter 9 concludes this and discusses possible future work.

Chapter 2

Fundamentals

This thesis is about access control in loosely coupled information systems. More explicitly, it explores access control and other security specifications and their enforcement for workflow specifications, i. e. specifications of service compositions, in service-oriented architectures (SOA). This chapter therefore first introduces the characteristics of service-oriented architectures (Section 2.1) and their technical realisation (Section 2.2). We then explain workflow management and its use for service compositions in SOA (Section 2.3). Finally, we describe security on different levels, ranging from basic concepts to technical solutions in SOA (Section 2.4).

2.1 Service-oriented Architectures

One of the oldest problems in software engineering is the decomposition of software and its functionality into smaller parts [Par72]. Back in 1972, Parnas saw the following advantages of modular programming: (1) Modules could be developed independently, leading to shorter development times. (2) The entire product (using Parnas's terminology) becomes more flexible, because it "should be possible to make drastic changes to one module without a need to change others." (3) The whole system becomes easier to understand because one can study one module at a time.

While these goals of modularisation are certainly important, the focus of *service-oriented architecture* (SOA) is slightly different. The term can best be explained by looking at its individual components:

- SOA is not a methodology for developing independent pieces. Instead, it defines an *architecture*, i. e. an abstract, standardised baseline explaining "the technology, boundaries, rules, limitations, and design characteristics that apply to all solutions based on this template" (cf. Section 4.3.1 of [Erl05]).

- SOA is based on individual units of logic known as *services* that "exist autonomously yet [are] not isolated from each other." They are "required to conform to a set of principles that allow them to evolve independently, while still maintaining a sufficient amount of commonality and standardization" (cf. Section 3.1.1 of [Erl05]).

- Finally, a service-oriented architecture is an architecture where automation logic is decomposed into services, and which encourages such a decomposition (cf. *ibid.*).

2.1.1 Principles of service-orientation

According to [Erl05] (Section 3.1.5), the principles of service-orientation address several design issues, such as:

- How should services be designed?
- How should service descriptions be designed?
- How should the relationship between services be designed?
- How should messages be designed?

Key aspects include:

- *Loose coupling*, i. e. a relationship that minimises dependencies.
- A *service contract*, i. e. a communications agreement that the service adheres to.
- *Autonomy*: services have control over their internal logic.
- *Abstraction*: the details of the logic of a service are hidden from the outside world; the service contract provides an abstract view only.
- *Reusability* as the goal behind the division of logic into services.
- *Composability*: A composite service can coordinate and assemble a number of other services.

These aspects are closely related and overlap. For example, reusability is a prerequisite for composability – to combine services in order to provide a more complex service, they have to be designed with that goal in mind.

Currently, the web services platform is the prevailing implementation of SOA principles. Data exchanges and message formats are based on open standards, as described in the next section. This promotes loose coupling and autonomy of services. While the core standards of the web services platform are stable, more advanced protocols are still maturing. Section 2.2 explains the core specifications, which cover service invocations and descriptions. Subsection 2.3.2 introduces WS-BPEL, used for workflows based on web services, and Subsection 2.4.4 and Subsection 2.4.5.3 cover security in service-oriented architectures and federated identity management, respectively.

2.2 The Web Services Platform

Service-oriented architecture, as explained in the previous section, is a conceptual model. Its usual implementation is based on web services, a widely accepted technology platform based on open standards.

```
<order xmlns="http://example.org/bookOrder">
 <books>
   <book pages="55">
     <title>Through the Looking Glass</title>
     <authors>
       <author>Lewis Carroll</author>
     </authors>
   </book>
   <book pages="102">
     <title>Moby Dick</title>
     <authors>
       <author>Herman Melville</author>
     </authors>
   </book>
 </books>
</order>
```

Figure 2.1: Basic XML example

2.2.1 XML and XML Schema

eXtensible Markup Language (XML) [BPM+08] is a text-based representation for semi-structured data. The structure of an XML document is made up of nested *elements* enclosed in *start* and *end tags*. Elements can have attributes, which are stated in the start tag, and text or other elements as children. Figure 2.1 shows a very basic example of an XML document. For a detailed description of features such as comments, processing instructions, escaping, etc., the reader is referred to the specification. Note that the `order` element in the example has an attribute called `xmlns`, which specifies the default *namespace* for the `order` element and its children. Namespaces [BTT+09] are an XML extension that allows less ambiguous element (and attribute) names, thereby increasing interoperability. With this extension, names can be *qualified* with a namespace, avoiding name clashes when combining different XML-based data formats. This is important for specifications such as SOAP or WSDL, as we will see below.

It is evident that this example follows a fixed structure. The purpose of *XML Schema* ([WF04, TMBM04, MB04]) is to define the structure of an XML document and allow automated checks of whether a document adheres to this definition. With XML Schema, one can specify which elements may be nested, which attributes may occur, and which data types are allowed. The schema in Figure 2.2 matches the example from Figure 2.1.

2.2.2 SOAP Messages

SOAP[1] ([ML07, GHM+07, MGH+07]) is the messaging framework used by web services. SOAP messages are XML documents. They consist of an `Envelope`

[1]Originally, *SOAP* was an acronym for "Simple Object Access Protocol". This is no longer the case in the current version; "SOAP" is now considered a standalone term and appears as-is in the titles of the respective specifications.

```
<?xml version="1.0" encoding="utf-8" ?>

<xs:schema elementFormDefault="qualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://example.org/bookOrder">

  <xs:complexType name="AuthorsType">
    <xs:sequence>
      <xs:element name="author" type="xs:string" minOccurs="1"
                  maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="BookType">
    <xs:sequence>
      <xs:element name="title" type="xs:string" />
      <xs:element name="authors" type="AuthorsType" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="ExtendedBookType">
    <xs:complexType>
     <xs:complexContent mixed="false">
       <xs:extension base="BookType">
         <xs:attribute name="pages" type="xs:integer" />
       </xs:extension>
     </xs:complexContent>
    <xs:complexType>
  </xs:complexType>

  <xs:element name="order">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="books">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="book" type="ExtendedBookType" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="orderStatus" type="string" />

</xs:schema>
```

Figure 2.2: XML Schema for the document from Figure 2.1

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Body>
   <order xmlns="http://example.org/bookOrder">
     <books>
       <book pages="55">
         <title>Through the Looking Glass</title>
           <authors>
             <author>Lewis Carroll</author>
           </authors>
       </book>
       <book pages="102">
         <title>Moby Dick</title>
         <authors>
           <author>Herman Melville</author>
         </authors>
       </book>
     </books>
   </order>
 </env:Body>
</env:Envelope>
```

Figure 2.3: Example of a SOAP Message

element, which can contain a `Header` element and must contain a `Body` element. Figure 2.3 shows a very simple example. To signal error conditions, SOAP provides a special `Fault` element that can be used as the content of the `Body` instead of payload.

The SOAP specification provides an encoding scheme for remote procedure calls. The next subsection introduces WSDL, which is used to describe the messages sent to and received from web services. SOAP also defines a processing model for SOAP messages and bindings to underlying transport mechanisms.

2.2.3 Service Descriptions (WSDL)

The Web Services Description Language (WSDL) [CCMe01][2] makes it possible to describe the interfaces of service endpoints.

A WSDL definition consists of two parts: an abstract description of the service interface, and a concrete description specifying where to find the service and how to access it. The abstract description starts with the types used, expressed as an XML Schema definition[3]. Next are definitions of messages, which can consist of a number of parts, each of a specific type. Port types

[2]The presentation here is based on version 1.1 of the language. Version 2.0 has been published as a W3C Recommendation ([Le07, CWMe07, LHC+07, Ved07]). However, it has not yet been picked up by the technical community. For example, WS-BPEL 2.0 (introduced later in this thesis) and the Java API for XML-Based Web Services (JAX-WS) 2.0 [CHe06] are both based on WSDL 1.1.

[3]WSDL recommends the use of XML schema to ensure interoperability, but other type systems are permitted in principle.

consist of one or more operations; each operation defines an input message, output message, and possibly a fault message. In the concrete description, a binding specifies a transport mechanism (commonly, SOAP over HTTP) for each port type, and the encoding to be used for each operation of that port type. Ports specify the location of a concrete instance of a port type with a previously defined binding. Finally, a service is defined as a bundle of ports.

2.2.4 Service Discovery

Because loose coupling is one of the key principles of SOA, the services used in an application often are not yet known when the application is created. As a result, it is necessary to discover services at runtime and bind them to the application. On the one hand, these services have to offer the functionality required by the application. On the other hand, non-functional properties such as quality of service or trustworthiness are also important.

The Universal Description, Discovery and Integration specification (UDDI) defined a protocol for web service registries, including formats for meta data about services. ID-WSF, which we will introduce in Subsection 2.4.5.3, does not support the same kind of complex service descriptions, but allows discovery of services based on a user's identity. An issue important in practice is the selection of services based on non-functional properties, including security properties. [BEdH+10] describes different ways to determine the trustworthiness of a service in order to take it into account for service selection. These use XACML (see Subsection 2.4.3.3) as the protocol of their Trust PDP, but allow the subject (i. e. the service) to be omitted, so that it becomes possible to *discover* services that are trustworthy according to the applicable trust policy.

2.3 Workflow Management

Workflow management is an interdisciplinary field with relations to business administration, formal methods and software development. This section provides a brief introduction, based on [Wes07].

A *business process* consists of activities performed in coordination to achieve a business goal. Activities in business processes can be performed manually by employees or with the help of information systems. A *workflow* is the automation of a business process, in whole or in part. During a workflow, information and tasks are passed between participants according to fixed rules. A workflow management system (WfMS) is a generic software system driven by explicit process representations (i. e. workflows) to coordinate how these processes are performed. It can interact with workflow participants and invoke tools and applications. It can thus be used to integrate different applications and improve collaboration between knowledge workers. A *workflow model* consists of activity models and execution constraints between them. A *workflow instance* represents a concrete case of a workflow, consisting of activity instances.

Business processes are identified by involving stakeholders in the organisation, resulting in workflows usually modelled using a graphical notation. The workflow models are then implemented, adding technical configuration. Several approaches exist to add the necessary configuration directly to the graphical workflow models [WS07, MTM09, RFMP07]. These approaches use

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookOrder"
    targetNamespace="http://example.org/bookOrder"
    xmlns:tns="http://example.org/bookOrder"
    xmlns:xsd1="http://example.org/bookOrder"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <!-- schema element from the example above -->
  </types>

  <message name="BookOrderInput">
    <part name="body" element="xsd1:order"/>
  </message>

  <message name="BookOrderOutput">
    <part name="body" element="xsd1:orderStatus"/>
  </message>

  <portType name="BookOrderPortType">
    <operation name="BookOrder">
      <input message="tns:BookOrderInput"/>
      <output message="tns:BookOrderOutput"/>
    </operation>
  </portType>

  <binding name="StockQuoteSoapBinding" type="tns:BookOrderPortType">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="BookOrder">
      <soap:operation soapAction="http://example.com/BookOrder"/>
        <input>
          <soap:body use="literal"/>
        </input>
        <output>
          <soap:body use="literal"/>
        </output>
    </operation>
  </binding>

  <service name="BookOrderService">
    <documentation>Exemplary BookOrder Service</documentation>
    <port name="BookOrderPort" binding="tns:StockQuoteSoapBinding">
      <soap:address location="http://example.com/bookorder"/>
    </port>
  </service>

</definitions>
```

Figure 2.4: Example of a WSDL document

so-called annotations, which usually specify security goals on varying levels of abstraction. The annotations refer to entities that are represented by graphical elements. Examples of such elements include actors (users or services), represented by grouping activities they are responsible for, data flow, represented by message edges, and activities, which constitute the basic elements of a workflow. The meaning of such an annotation is that the security goal is important for the entity in question. For example, the entity has to be accessed or processed in a specific way, or a specific instance of the entity at runtime has to meet certain criteria. To enforce the security requirements expressed by such annotations, they are either translated into policies evaluated at runtime, or code is generated to that end, e. g. by inserting predefined workflow fragments. This turns workflow management into a form of model-driven development.

For modelling workflows, the *Business Process Modelling Notation* (BPMN) is a de-facto standard [Obj08]. A brief explanation of the elements of BPMN is helpful to understand the diagrams in Appendix A. Circles represent events. In particular, circles with a normal stroke width represent start events, while bold circles stand for end events. In the basic case, a diagram contains one start and one end event. Rounded rectangles represent activities. Gateways, i. e. forks or joins in the workflow's control flow, are represented by diamonds. The symbol in the diamond indicates the type of gateway. A 'plus' symbol indicates parallel execution and a 'cross' symbol indicates alternative execution. Solid edges running mostly horizontally represent the control flow. Diagrams are structured into multiple containers stacked vertically, which are called *pools*. Each pool represents one actor or organisations's view of the workflow. It contains events, activities and gateways connected by control-flow edges. Pools can be further divided vertically into *lanes*. Their meaning is not fixed by the standard. A common use of lanes is the representation of different roles within an organisation. Activities in different pools can communicate by exchanging messages. This data flow is represented by dashed edges running mostly vertically.

2.3.1 The Workflow Reference Model

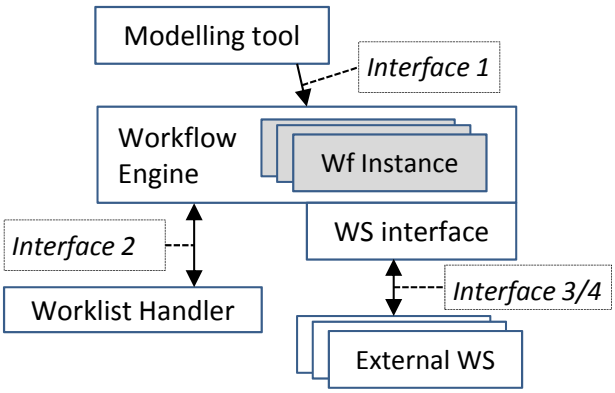


Figure 2.5: Basic Architecture of a WfMS (WfMC Reference Model)

Workflows orchestrate the behaviour of services and humans. To this end, they define a control flow and a data flow between the entities involved, based on message passing. The reference model of the Workflow Management Coalition (WfMC) [Hol95] defines an architecture for WfMS that offers this functionality (see Figure 2.5). The central component is the workflow execution engine, which has several interfaces. Interface 1 is between the engine and the workflow modelling tool, and facilitates the deployment of workflow definitions, including their security configurations. Interface 2 connects the worklist handler to the engine. The worklist handler performs interactions of the WfMS with users via *human tasks*. Note that there is usually a single worklist handler for all users. Users can choose from different tasks, which are often available to different users, but ultimately only performed by one user. Interfaces 3 and 4 connect the engine to applications and other WfMSs respectively. These interfaces are very similar at the technical level, and the distinction is mainly due to historical reasons. Interface 5 deals with administration and monitoring by providing audit data to an external component and allowing a management tool access to the engine. In SOA, workflow definitions commonly use WS-BPEL [JE07]. Applications and other workflows are provided as web services using SOAP [GHM+07].

2.3.2 Executable Workflows

For execution, another language is used[4]. The *Web Services Business Process Execution Language for Web Services* (WS-BPEL, or BPEL for short) [JE07] focusses on workflows in service-oriented architectures, i. e. compositions of web services.

Because BPEL builds upon web services, workflow definitions include WSDL documents. These are abstract in the sense that they do not include `<binding>` or `<service>` elements. In addition to the usual WSDL definition, they contain partner link types. Each partner link has a name and either one or two roles that define the port types the communication partners must provide. In the workflow definition itself, partner links are defined. They refer to partner link types and specify the role assumed by the workflow and/or its communication partner. BPEL allows variables to be specified using either a WSDL message type, an XML Schema type, or an XML Schema element definition. A BPEL workflow consists of a single (usually complex) activity. Simple activities, for example, perform communication (`<receive>`, `<reply>`, `<invoke>`) or copy variable values (`<assign>`). Complex activities provide control flow for the activity or activities they contain (e. g. `<sequence>`, `<if>`, `<while>`, `<flow>`). Other definitions include event and fault handlers. Correlation sets are an important mechanism allowing messages to be routed to workflow instances based on parts of their application-specific payload.

BPEL workflows are executed by dedicated middleware components referred to as *workflow engines*. These engines are usually embedded in a web-service stack that may add support for protocols facilitating functionality such as message encryption or reliability. The level of abstraction used by BPEL

[4]Version 2 of BPMN, now standing for *Business Process Model and Notation* [Obj11], specifies execution semantics, but is not yet commonly used.

also allows for the specification of more concrete and application-specific non-functional properties, such as security.

2.3.3 Workflow Aspects

Curtis et al. [CKO92] have described common perspectives on workflow, namely the *functional*, the *behavioural*, the *organisational* and the *informational* perspective. Jablonski and Bussler [JB96] used a slightly different terminology and structure. They add the *operation* perspective, and differentiate the behavioural perspective into the actual control flow and the causality perspective. They also add some perspectives that are cross-cutting in nature, in particular security and history. In the requirements analysis (Section 5.1), we follow the structure introduced by Charfi and Mezini [CM06], which comprises the following different perspectives, also known as aspects: The functional perspective describes the workflow elements that can be performed, i. e. tasks and sub-workflows, and their connection with informational entities (data objects). The behavioural perspective deals with the control flow, including temporal and causal relationships. The organisational perspective represents the relationship of the workflow to organisational structures and who performs operations. Finally, the operational perspective focuses on the involvement of tools and applications in the workflow.

2.4 Security

2.4.1 Different Definitions of Security

What security actually means is quite fuzzy in the literature. In order to get a complete picture, this section summarises several sources: two established textbooks and an international standard.

2.4.1.1 Whitman and Mattord

Whitman and Mattord [WM07] define security as "the quality or state of being secure – to be free from danger." They identify different areas of security, namely physical, personal, operations, communications, network and information security.

They see information security as a combination of network security, computer and data security, and information security management. Their book focusses on the latter aspect.

The authors identify critical characteristics of information, which constitute its value and thus need to be protected:

- *Availability* "enables authorised users [...] to access information without interference or obstruction.".
- *Accuracy* means that information is "free from mistakes or errors."
- *Authenticity* "is the quality or state of being genuine or original."
- *Confidentiality* "ensures that *only* those with the right and privileges to access information are able to do so."

- *Integrity* is the property of being "whole, complete, and uncorrupted."
- *Utility* "is the quality or state or having value for some purpose or end."
- *Possession* means "having ownership or control of some object or item."

Information systems comprise software, hardware, data and people. To ensure security not only of information itself, but also of the systems that use, store and transmit it, these must be protected as well.

2.4.1.2 Pfleeger and Pfleeger

For Pfleeger and Pfleeger [PP02], security is about protecting valuable (computer-related) assets against attacks. They define security in terms of three goals: confidentiality, integrity and availability. This is a less fine-grained version of the categorisation used by Whitman and Mattord. For example, integrity includes accuracy, authenticity and utility, and availability includes utility as well.

2.4.1.3 ITU Standards

The International Telecommunication Union (ITU)'s Telecommunication Standardization Sector[5]. has created a security framework for data networks.

X.800 [ITU91] introduces the general structure: *security services* for different security goals, namely authentication, access control, data confidentiality, data integrity and non-repudiation. These services are implemented by specific *security mechanisms*, such as encipherment to ensure confidentiality. X.800 also recognises the need for management functionality such as collection of security audit information and key management.

X.810 [ITU95c] gives an overview of several security frameworks and defines frameworks for authentication, access control, non-repudiation, confidentiality, integrity, security auditing and alarms, and key management.

2.4.1.4 Summary

Security can be seen as the protection of some entities against threats. The categorisations of security vary considerably. Both textbooks classify important properties entities can have. The difference between them is that the partitioning of Whitman and Mattord is more fine-grained than that of Pfleeger and Pfleeger. By contrast, the X.800 framework classifies the mechanisms used to achieve this protection. While some mechanisms can be mapped one-to-one to a property (e. g. authentication ensures authenticity), this is not possible for others.

2.4.2 Approaches to Security

But how does one achieve security goals like these? This subsection outlines the overall analysis and design process leading to a secure system, based on [PP02].

[5]Before 1993, ITU-T was known as the International Telegraph and Telephone Consultative Committee (CCITT, from the French *Comité Consultatif International Téléphonique et Télégraphique*). Thus, older standards still bear the CCITT label.

Security is about protecting valuable assets of a computing system, such as its hardware, software and data. A chain is only as strong as its weakest link, and the same applies to security in computing, known as the *Principle of Easiest Penetration*: An attacker can use any available means of penetration. Therefore, instead of pursuing attacks against which a strong defence has been installed, the attacker can go for the "weakest link in the chain" and use attacks which had not been considered when designing protections, and against which no or only weak defences are in place.

Risks to the security of a system are opened up by *vulnerabilities*, i. e. weaknesses in the security system. A *threat* is "a set of circumstances that has the potential to cause loss or harm" by triggering a vulnerability. An attack is the exploitation of a vulnerability, originating from either a human or another system. Dealing with threats requires implementing a *control* for the respective vulnerabilities. Possible controls are software controls, hardware controls, policies and procedures, and physical controls. The number of possible threats is uncountable, and implementing a control has a certain cost (such as time for development and maintenance) associated with it. Moreover, a control can introduce new vulnerabilities, and also create hidden costs by decreasing the availability of assets, thereby impeding the normal operation of the system. This means that controls must be justified by a significant *risk*, i. e. the possibility for harm to occur, which happens when a threat against a vulnerability is realised. Security must be easy to use, otherwise it will not be used at all or deliberately circumvented[6].

In conclusion, an appropriate balance between protection and availability is necessary, and security mechanisms must be as unobtrusive and easy-to-use as possible. Furthermore, security enforcement should be able to rely on building blocks that are known to work well, specifically in the case of concrete measures necessary to enforce a certain security goal for certain kinds of entities. When these building blocks use concepts that are sufficiently abstract, using them is straightforward. In a workflow context, it is necessary to provide mechanisms enforcing security goals for the different entities involved in a workflow.

2.4.3 Access Control

Access control is the one of the most diverse areas of security. It is also hard to manage, and many approaches have been proposed. This subsection thus introduces some of the basic concepts important in workflow management and service-oriented architectures. The purpose of access control is to control attempts by users to perform certain actions on objects (e. g. reading the system configuration). In general, access control can happen on various levels of abstractions, depending on the available access mechanisms.

2.4.3.1 Basic Access-control Mechanisms

One basic mechanism is *discretionary access control* (DAC), which can be seen as a function $(U, O, A) \to \mathbb{B}$ which determines whether a user $u \in U$ can perform the action $a \in A$ on an object $o \in O$. Different implementations of DAC are possible, such as access control lists or an access control matrix.

[6] See Chapter 17 of [Sch00] for examples.

However, DAC becomes hard to manage with large numbers of users and objects because the access control decision must be explicitly coded for each combination. Therefore, Ferraiolo and Kuhn [FK92] introduced *role-based access control* (RBAC). RBAC avoids this explosion of combinations by introducing an abstraction between users and objects, namely roles. Permissions on objects are defined in terms of roles, and each user in turn is assigned one or more roles. They are granted access to an object if they hold one or more of the necessary roles. Management of the permission system is also easier, as roles are coupled to the organisational structure (e.g. job positions). By now, RBAC has become a formal standard in the United States [ANSI04]. The RBAC reference model includes users, roles, permissions and sessions. Permissions are assigned to roles, and users are assigned to one or more roles. Users enable roles in the context of a session and then hold the permissions assigned to these roles for the duration of the session. Extensions are addressed below.

Attribute-based access control (ABAC) is a generalisation of RBAC. The entities involved in an authorisation decision, including subjects and resources, are characterised by attributes. Attributes are name/value pairs and can be issued by different organisations. Policy rules are expressed in terms of attributes. The most prominent example of an ABAC-based language for authorisation policies is XACML, described in Subsection 2.4.3.3.

2.4.3.2 Constraints

Separation of duty prevents fraud and abuse of authority by distributing the privileges needed to accomplish a certain result among several actors. [KF95] describes several types of *separation of duty* (SoD) constraints for the RBAC model. The simplest type is static SoD, which is based upon mutually exclusive roles. Static SoD is enforced when roles are assigned to users, i. e. during design. No user may be assigned multiple roles designated as mutually exclusive. Dynamic SoD in turn requires a mechanism by which users can activate roles. They can only perform operations allowed for their current active roles. Dynamic SoD prevents the simultaneous activation of mutually exclusive roles. These two types of SoD are included in the RBAC standard [ANSI04].

Several similar definitions of constraints have been proposed in the literature. Gligor et al. [GGF98] provide a formalisation of these and other types of SoD. Ahn and Sandu [AS99] present a language that allows the definition of separation of duty constraints. Chadwick et al.[CXO+07] present a model called *Multi-session Separation of Duties* (MSoD). MSoD constraints do not depend on user sessions but apply to business contexts, i. e. a set of business processes for which an MSoD policy is valid. The authors define multi-session mutually exclusive roles and mutually exclusive privileges through cardinality constraints. This means that for a specified set of roles or privileges respectively, the constraint limits the number of roles a user may activate or privileges he may exercise to a given maximum value.

Tan et al. [TCG04] define constraints on workflow executions. In their analysis, they consider the possibility that the same task is instantiated more than once during a workflow execution, i.e. the same permission is used more than once. This is they reason why their definition of entailment constraints and cardinality constraints is different from [ANSI04]. According to their definition, entailment constraints restrict who is allowed to perform a task t_1 based

on who has performed another task t_2. Examples are separation of duty and binding of duty (i. e. the same user has to perform t_1 and t_2). Cardinality constraints concern tasks which have to be performed more than once during a workflow execution, e. g. due to loops. They define the *consistency* of such constraints, i. e. whether the workflow is still executable, and develop an algorithm to check the consistency. Basin et al. [BBK11] have proposed a solution for the problem that the semantics of authorisation constraints are not precise when a workflow contains loops: They place so-called *release points* for constraints in workflow definitions. When the execution of a workflow reaches a release point, the constraint specified for future task executions is effectively as if none of the tasks involved in the constraints had been executed previously. Burri and Karjoth [BK12] have defined additional constraints following the same basic pattern.

2.4.3.3 XACML

Extensible Access Control Markup Language (XACML) [Mos05, Ris13] allows authorisation policies to be specified. These policies specify whether a given subject may perform a given action on a given resource, possibly taking the environment into account. They are based on attributes for subjects, resources, actions and the environment. XACML defines a (non-normative) data flow model for access control systems. It contains components that grant or deny access requests (policy enforcement point, PEP), take the actual decision (policy decision point, PDP), create policies (policy administration point, PAP) and provide context information for the evaluation of policy decision requests (context handler). The context information includes attributes of subjects, resources and the environment which are collected by the policy information point (PIP), from which the context handler can retrieve them. In addition, XACML defines a format for decision requests and results. An RBAC profile [And05, Ris14b][7] describes how to encode role-based authorisation policies using XACML. This profile also supports role hierarchies, but it does not currently support separation of duty.

2.4.4 Security in Service-Oriented Architectures

The textbook by Kanneganti and Chodavarapu [KC08] addresses security with a focus on service-oriented architectures. The authors classify security into network security, platform security and application security. They see the strongest impact of the SOA paradigm on application security. The reason is that with SOA, network security and application security become similar, as the network takes over security tasks traditionally handled by applications.

They see similar "functional aspects" of security as in traditional applications, namely authentication, authorisation, data confidentiality, data integrity, protection against attacks, and privacy. In a SOA setting, several "non-functional aspects" of security come into play as well: Interoperability means making sure that adding security does not break compatibility. Manageability is important because a typical SOA comprises a large number of autonomous services. Another important aspect is ease of development, because otherwise security will not be adopted.

[7] The RBAC profile for XACML 3.0 [Ris14b] is not yet an official OASIS Standard.

The authors identify new security approaches in SOA environments: *Message-level* security allows more fine-grained protection than transport-level security. For example, parts of the message can be usable for parties relaying a message, while others are secured end-to-end. *Security as a service* allows security functionality to be encapsulated. This makes security easier to use for application services and allows it to evolve independently from application functionality.

WS-Security [LK06b] provides security on the message level. More specifically, it allows the encryption and signing of SOAP messages or parts of them. WS-Trust [Ant07] defines a protocol for handling security tokens. WS-SecurityPolicy [NGG+07] provides for declarative specifications of security requirements.

Kanneganti and Chodavarapu also describe common architectural patterns in SOA security. Security can be enforced on the endpoints by so-called handlers, or during transport using intermediaries. The latter alternative allows message routing to be influenced in line with security requirements. When using security services to implement the actual security functionality, these can be called explicitly by endpoints. It is also possible to call them implicitly, but this requires special support from the network. Generic security services are often configured using declarative policies, e. g. using WS-SecurityPolicy.

2.4.5 Identity Management

An important prerequisite to achieve security goals is to reliably manage the *identity* of entities involved. Identity is "a property of a subject that enables it to be identifiable and to link items of interest to the subject" [PH10]. In this context, the concept of *digital identity* is important. It refers to attribute values of an individual that are immediately accessible by technical means. Individuals expose different parts of their identity in different contexts. *Identity management* means managing these various partial identities.

Identity management is important for security, in particular for authentication and access control. An identity management system has to check the identifiers and attribute values that a subject claims to possess, i.e. authenticate the identity of the subject. On this basis, a system can decide whether to grant access to a particular resource.

In the SOA context, identity management has evolved into *federated identity management* (FIM), a set of technologies and processes that let computer systems distribute identity information dynamically and delegate identity management functionality to other systems [MR08]. An FIM infrastructure allows service providers to offload the cost of managing user attributes and login credentials to an identity provider, thereby increasing scalability. It also provides users with single sign-on (SSO), making it easier to use services from different providers [Cha09].

From a technical perspective, there are several competing frameworks specifying FIM functionality. The most important ones are Security Assertion Markup Language (SAML) [OAS08] and OpenID [Ope07]. The basic concepts of SAML and OpenID are similar, but the two technologies are used in different settings. OpenID is a lightweight protocol mainly used for user interactions on the Web. By contrast, SAML is more suited to service-oriented architectures using web-service technologies. We will therefore concentrate on SAML

and the technologies that extend it. The rest of this subsection presents FIM concepts and their implementation in the SAML framework.

2.4.5.1 FIM Concepts

Architecture and administration of a federated identity-management system The basic roles in an identity management federation are an *asserting party* (also called *identity provider, IdP*) which makes claims about individuals, and a *relying party* (also called *service provider*). The individual an identity provider makes claims about is called a *subject*. Governance tasks in FIM comprise (a) establishing the relationship between users and identity providers, i.e. creating accounts, verifying user identities by some real-world mechanism and providing credentials, and (b) establishing a federation between an identity provider and a service provider. This second group of tasks implies the creation of a trust relationship, i. e. defining which identity providers are trusted to correctly authenticate users. Technically, this requires exchanging configurations such as network addresses and public keys.

Types of identity information Identity information handled by an identity provider takes the form of *attributes*, which contain some statement about the user that may be relevant for the service provider (e. g. the user's driving license category). A special kind of attributes are *identifiers*. They uniquely identify a user in the context of one domain or system and are usually not valid outside of that system. It is also possible to have pseudonymous identifiers which are only valid for the current interaction of the user with a given system (service provider).

Transfer of identity information to service providers Regarding this topic, there are two orthogonal criteria:

First, one has to distinguish between scenarios involving direct interaction of the user with the service provider on the one hand, and scenarios where two service providers interact without the user being directly involved. Direct interactions of a user with the service provider usually happen via a web interface. The service provider initiates SSO by redirecting users to their identity provider; after the user has logged in there, the identitity provider then provides identity information relating to that user to the service provider. Interactions between service providers are based on on web service calls. In addition to the payload, identity information is included in the call. Possibly, the identity provider is involved to map between identifiers valid for the two involved service providers (cf. the explanation of ID-WSF in Subsection 2.4.5.3 below).

Second, different kinds of identity information are possible: Attributes (including identifiers), which have been asserted by an identity provider, can be transferred to a service provider. In addition, it is possible for the service provider to receive tokens that enable it to act on behalf of the user.

Services based on identity information Service providers use identity information mainly for authorisation, but also for other purposes such as customisation. In particular, identifiers allow recognising users throughout a multi-step interaction such as a booking process where the user repeatedly has to take

decisions and wait for external events. Some applications require separation of duty between different steps. For example, no user should be able to approve their own actions. Other identifiers can be used for customisation, such as adress data as the suggested shipping address.

An application can also use certain (non-identifying) attributes to provide different functionality to different groups of users. For example, a car-booking application might filter available cars depending on the customer's driving license category. This is a case of authorisation, because rental cars are only available to users holding a suitable driving license. The application could also use attributes like age to adjust the presentation by preferring offers deemed most interesting to the respective age group, while still allowing the user to choose other offers. Finally, applications can consist of building blocks that identity information is necessary or useful for. Examples include a travel agency's flight booking service, a calendar service that itinerary data is entered into, or a service that authorises payment based on the user's consent expressed through a more secure channel.

The *Identity Web Services Framework* (ID-WSF) [Lib06] specifies the notion of an *identity web service*, or *identity service* for short. This term is "an abstract notion of a web service that acts upon some resource to either retrieve information about an identity or identities, update information about an identity or identities, or perform some action for the benefit of some identity or identities." In other words, the functionality of an *identity web service* depends on whom it provides a service for. Examples of such functionality include: (a) Storing information about the holder of an identity (user). In this case, the service is able to answer requests for such information. (b) Interacting with the user and returning their decision, such as the authorisation of a payment. (c) Services that can take a decision based on instructions from a user. In the case of workflows, this includes a service that can declare the user's consent to terms of service based on a policy or on their choice on prior occasions.

Services often depend on information provided by other services. This is also true for identity web services. To achieve their functionality, they have to call other identity web services on behalf of the identity that invoked them, by including credentials.

Access control based on identity information In Subsection 2.4.3.3 we introduced XACML, a language designed for access control policies. XACML implements attribute-based access control. When used in a federated identity-management infrastructure, it is natural to rely on the attributes of subjects provided by this infrastructure. Attributes are asserted by the identity provider and have to be validated by the service provider before they are used for access control. In the reference data-flow model of XACML, validation would be part of the PIP's tasks. In PERMIS [ISS], a dedicated component called *credential validation service* is used. In order to enforce constraints such as separation of duty (Subsection 2.4.3.2), identifiers are needed. These are also provided by a federated identity-management infrastructure and validated in the same way as other attributes.

Reliability of identity information When service providers use identity information, especially for access control, they must be able to rely on it. How-

ever, the level of reliability required depends on the application. For example, for transactions dealing with small amounts of money, less secure authentication might suffice. Write access to a resource might require a higher level of reliability than read access. To allow service providers to assess whether identity information is acceptable for them, so-called *levels of assurance* have been specified. Levels of assurance are integer values, with Level 1 being the weakest and Level 4 the strongest. Chadwick [Cha09] suggests distinguishing between registered identity attributes and authoritative identity attributes.

2.4.5.2 Social and Privacy Considerations

Identity management is not only a technical issue. Dealing with digital identities also impacts social processes. In particular, it facilitates large-scale processing of personally identifiable information. Specific legal restrictions apply to such processing. A report by the Organisation for Economic Co-operation and Development (OECD) [Run08] adresses public policy issues arising from the increasing use of identity management methods.

The *laws of identity* [Cam06] address these and similar concerns. These are recommendations for FIM implementations and have resulted from intensive discussions within the identity management research community. Several laws concern privacy aspects of FIM features and associated user-control mechanisms.

One law requires the system to "only reveal information identifying a user with the user's consent" (Law 1). Another (Law 2) requires only "minimal disclosure for a constrained use." Another law requires "unidirectional" identifiers valid only for one service provider (Law 4), to prevent the combining of identity information provided to different parties. In workflow management, this means isolation between workflow instances. Finally, the identity system should provide "unambiguous human-machine communication mechanisms offering protection against identity attacks" (Law 6). The user experience should be consistent in different situations (Law 7). The remaining laws (Laws 3 and 5) do not address privacy features of FIM systems, but compliance and technical interoperability in FIM systems. The laws of identity are desirable privacy properties for any FIM system, including an FIM-enabled workflow management system.

General privacy protection principles are also relevant to the field of identity management. Important sources for such principles are the OECD privacy protection guidelines [OEC13] and the EU Data Protection Directive [EC95]. The OECD guidelines comprise *Collection Limitation*, *Data Quality*, *Purpose Specification*, *Use Limitation*, *Security Safeguards*, *Openness* (e.g. about practices employed), *Individual Participation* (e.g. a right to information and to demand correction of data) and *Accountability*. The EU Directive states *principles relating to data quality* (Article 6) and defines *criteria for making data processing legitimate* (Article 7), *the data subject's right of access to data* (Article 12), *the data subject's right to object* (Article 14) and rules for *security of processing* (Article 17).

2.4.5.3 Technical Specifications for Federated Identity Management

We will now briefly introduce technial specifications that implement the concepts from the previous subsection. As already explained, we will only address SAML and specifications that build on it. This should also be useful as a reading guide to these specifications.

SAML The fundamental data structures in SAML are assertions [CKPM05], which "carry statements about a principal that an asserting party claims to be true". The specification defines different kinds of statements. Authentication statements assert that the user has been authenticated and contain the means and time of the authentication. An attribute statement gives an attribute type and value claimed to be true for the user. Identifiers are given in the `Subject` element of the assertion, which contains a `NameID` element. [CKPM05] defines different kinds of `NameID`s. Core alternatives are globally unique names like e-mail addresses and X.509 subject names, privacy-preserving persistent identifiers, which have no correspondence to an actual identifier and are specific to a given service provider, and transient identifiers, i. e. random and temporary values.

[CKPM05] also defines protocols, i. e. exchanges of protocol messages, which contain assertions in most cases. Examples include the *Assertion Query and Request Protocol* and the *Authentication Request Protocol*. Note that protocols only define messages, while bindings specify a particular transport mechanism such as HTTP POST. Profiles [HCH+05] define in greater detail how to use SAML for a particular application, providing for better interoperability between different implementations. There are two types of profiles. One contains a set of rules describing how to embed SAML assertions into or extract them from a framework or protocol. The other describes how to use SAML features in a particular context by specifying further details left open in the core specification. An important example is the *Web Browser SSO Profile*, which implements the Authentication Request Protocol using the HTTP Redirect, HTTP POST and HTTP Artifact bindings. It allows identity information to be transferred from an identity provider to the web frontend of a service provider. The SAML Token Profile of WS-Security [LK06a] allows SAML assertions to be used as tokens in WS-Security [LK06b] headers of SOAP messages. The service provider receiving the message has to validate the evidence provided by the caller according to the confirmation method specified in the assertion (e. g. that the sender holds a specific key). These two profiles accordingly support the two major ways of providing identity information to service providers, namely through user interactions on a web frontend and through web-service calls.

[KCM+05] defines how to express so-called *authentication context*, i. e. additional information for assertions that allows service providers "to assess the level of confidence they can place in that assertion." The specification distinguishes five dimensions relevant for the reliability of identity information: (1) Identification, i. e. how the original relationship between the subject and its identifier was created, (2) Technical protection, i. e. how the secret allowing the subject's authentication is kept secure, (3) Operational protection, i. e. security procedures in place at the identity provider, (4) Authentication method,

i. e. characteristics of the authentication mechanism used (such as password vs. smartcard), and finally (5) Governing agreements, i. e. the underlying legal framework, such as liability constraints and contractual obligations. The specification allows detailed stipulation of the technical or organisational measures taken. However, it leaves it to implementations to define requirements for the authentication context or compute an aggregate value such as the level of assurance mentioned above.

XACML, a language for access control policies, has already been introduced above. The SAML 2.0 Profile of XACML 2.0 [AL05, Ris14a][8] defines an extension of XACML authorisation decision queries so that SAML attributes can be transmitted to the policy decision point. This allows identity information to be used directly for authorisation.

ID-WSF The Liberty Alliance has developed the *Identity Web Service Framework* (ID-WSF) [JT06, Lan03]. ID-WSF uses SAML and related technologies to facilitate identity-aware service compositions. It has several capabilities that go beyond those of SOAP described in the previous section. In particular, ID-WSF allows services to be discovered based on the user's identity and provides a privacy-friendly mechanism for service invocation. ID-WSF also specifies in greater detail some functionality already defined by SOAP, such as SSO.

ID-WSF comprises several related specifications: The SOAP binding specification [HKA+07] defines how to invoke identity services. It defines technical details such as SOAP headers and status codes. The ID-WSF Security Mechanisms specification [Hir06b] defines the use of tokens for message authentication, including tokens that specify the invoking identity. [Hir06a] specifies how to use SAML assertions as authentication tokens. In addition, ID-WSF defines an *SSO Service* that allows a system to obtain SAML assertions as security tokens. It also defines an *Identity Mapping Service* that translates references to users into alternative formats or identifier namespaces [HAMC07]. The Discovery Service specification [CC07] defines a data format to describe (identity) web services and specifies a service that detects services of a certain type available to a given identity.

2.5 Summary

This chapter has introduced concepts and technologies from different fields relevant to the topic of this thesis. We have limited this treatment to the sound fundaments necessary to a basic understanding of each field. This excludes more complex concepts described in the literature, which we have yet to evaluate the usefulness of in relation to solving our research question. We will address this in the next chapter.

[8] The new version of this profile [Ris14a] is not yet an official OASIS Standard.

Chapter 3

Related Work

In this chapter, we now present some approaches that address problems similar to ours or parts of it. The explanations go deeper than the general overview in the previous chapter. Moreover, some of the approaches presented provide solutions for more concrete research questions instead of just a general framework. However, the approaches do not necessarily amount to complete solutions. Nevertheless, they can provide ideas for solving our research question, or a baseline for comparison.

3.1 Policy Specifications for SOA

The WS-Policy specification [YHV+07] provides a generic model for policies of entities in a web-service-based system. It also provides a syntax to express such policies. Policies are structured into policy alternatives, which in turn are collections of policy assertions. Policy assertions are application-specific and represent "a requirement, capability, or other property of a behaviour" of an entity. [VYO+07] provides a way to attach such policies to WSDL service descriptions. WS-SecurityPolicy [NGG+07] uses the generic framework provided by WS-Policy. It defines policy assertions that describe the encryption and signature mechanisms supported or required by web services. It supports different protocols, including WS-Security.

We already rely on the security mechanisms provided by ID-WSF for confidentiality and non-repudiation. The policies we require concern access control, which is not addressed by WS-SecurityPolicy.

3.2 Access Control Models for Dynamic Environments and Workflows

3.2.1 Team-Based Access Control

Thomas [Tho97] introduces the *Team-Based Access Control* model (TMAC). It applies to collaboration scenarios where several team members are assigned to a case and have to access data related to this case. The idea is to assign users and objects to a team at runtime. Users have a certain role in the team, and objects are of a certain type. Permissions are defined based on team roles and object types. At runtime, these abstract permissions are instantiated, so

that the holder of a team role gets access to objects of a certain type. For example, a rule grants a doctor assigned to a case access to each patient record related to the case. In TMAC, two team members with the same role in the organisation but performing different tasks in the case get the same permissions. Differences between the permissions of team members only result from the roles they already possess outside the context of the team. In [GMPT01], Georgiadis et al. extend this model with *context* (e.g. locations or time intervals) linked to a team, and describe more formally how a user joins teams and how the resulting permissions are computed.

The underlying ideas influenced our concept for permissions delegated to workflow participants (Subsection 8.2.3). TMAC itself is not directly applicable to workflows because it does not take workflow execution and user-task assignment in workflows into account (see also Subsection 5.1.4.3).

3.2.2 Workflow Authorization Model

Atluri et al. [AH96] were the first to address synchronisation between the control flow of a workflow and authorisation for data processed. They present the *Workflow Authorization Model* (WAM) that grants permissions for data processed in a task to the person executing it, based on so-called authorisation templates. An extension supports roles and authorisation constraints such as separation of duty. The authors also provide a Petri-net-based method to determine the possible authorisation states [AH00].

At first glance, WAM addresses requirements similar to ours. However, it directly specifies the subjects (users or programs) that have to perform a task, reducing its flexibility. WAM cannot express complex control flow using branches or loops, which are possible in WS-BPEL. It specifies the types of objects used and created in a task. Authorisation templates refer to object types and enable permissions only for the objects directly involved in a task. In other words, WAM is based on a data-centric workflow model, rather than business workflows where the control flow of workflows plays a major role and data items have a relationship to the workflow as a whole (see also Subsection 5.1.4.3). By contrast, our approach, our approach in Subsection 8.2.3 allows delegations of permissions valid for more than one task, and determines the delegate based on the user-task assignment of the workflow.

3.2.3 Task-based Authorization Controls

Thomas et al. [TS98] present an approach called *Task-based Authorization Controls* (TBAC). The goal of this approach is to "models access controls from a task-oriented perspective." Access control permissions are bound to authorisation, which here means an explicit act of granting permissions. TBAC introduces *authorisation steps*, which enable a predefined set of permissions, called the authorisation step's *protection state*. These permissions are associated with a usage count, thus eventually becoming invalid. Further, the authors specify the possible states of authorisation steps (e.g. 'dormant', 'valid' or 'hold') and the transitions between these states. TBAC also allows *composite authorisations*, which require approval by several actors.

TBAC is specified in a very abstract way. The specification is not detailed enough for an implementation. The predefined permissions are static, i. e. not

derived from some runtime information source such as users or data objects involved in a workflow. In our architecture, explicit approvals to perform a task are not mandatory (but described in Subsection 8.2.6). According to our approach in Subsection 8.2.3, permissions delegated to a workflow participant are instead based on these participants' assignment to certain tasks. A usage count for delegated permissions would be a possible extension. This would have to be part of the specification of required permissions. However, we have no requirement for such a feature, which is also not workflow-specific.

3.2.4 W-RBAC

Wainer et al.'s W-RBAC model [WBK03] uses RBAC extended with a concept of organisations and cases (i. e. workflow instances). It supports constraints such as separation of duty or binding of duty through predicates. Constraints ordered by priority are a key feature of W-RBAC. Users may override constraints when a workflow could not continue otherwise.

This feature could augment our architecture's support for constraints. However, none of our use cases require this feature.

3.2.5 Summary

The different approaches explained in the previous subsections differ in the context they use for determining which subjects are granted permissions on which resources, and regarding the significance of (workflow) tasks. All of them have in common that the consideration of tasks is not sufficient and that the way subjects and resources are determined does not fit our requirements.

3.3 Delegation

In environments where several users cooperate, it is necessary from time to time to transfer privileges and responsibilities from one user to another. This process is known as *delegation*. In the following, we present existing approaches to this topic. We can broadly classify these into two groups. First, one needs to decide under which circumstances delegation should be allowed. Second, one needs to make sure that delegation actually takes effect, i. e. perform delegation technically. Both aspects of delegation (delegation models and delegation mechanisms) are also relevant for the implementation of advanced access control models.

Delegation models Barka and Sandhu [BS00] extend RBAC by adding delegation, defining characteristics useful for the classification of delegation operations. (1) *Permanence* refers to the time duration of the delegation, which can be either permanent or temporary. (2) *Monotonicity* decides whether the delegating user retains the rights they have delegated. (3) *Totality* refers to whether a user has to delegate all permissions attached to a role, or can limit the delegation to a subset of those permissions. (4) *Administration* of the delegation can be done by the user themselves, or by an agent on their behalf. (5) The *level of delegation* can be unbounded or limited. For instance, single-step delegation does not allow any further delegation. (6) *Multiple delegation*

means that a user may delegate to more than one delegate at the same time. (7) *Agreement* refers to how delegation is agreed between delegator and delegate. It can either be unilateral (only agreed to by the delegator) or bilateral (agreed to by both delegator and delegate). (8) *Revocation* is another important issue, which can have unexpected effects. The authors describe several delegation models and their characteristics according to this taxonomy. Later, they describe an extension [BS04] with specific emphasis on role hierarchies. Zhang et al. [ZAC03] provide a framework for the delegation of RBAC roles, relying heavily on the taxonomy introduced by Barka and Sandhu. Atluri and Warner [AW05] address the delegation of tasks in workflow systems. They consider delegation constraints based on time, workload, and workflow variables. They analyse how to assign users to tasks so that all constraints are fulfilled. Wainer and Kumar [WK05] define a delegation model for RBAC that uses RBAC to represent delegation rights. These delegation rights can include conditions on the delegate, and limit the length of the delegation chain.

Delegation mechanisms Different ways exist to implement delegation technically. The main distinction is between policy- and token-based solutions. [TAS09b] provides a more fine-grained classification of ways for a repository user to provide third parties with access to repository content. This scenario is useful in a workflow context as well. The classification comprises the following cases:

1. Direct management of the repository's policy by the user,
2. issuing of authorisation tokens by the user,
3. issuing authorisation tokens to third parties on the user's request,
4. evaluating "sticky policies" attached to the repository contents by the user,
5. using a trusted delegation service that issues tokens on behalf of either the repository or the user.

The options differ, inter alia, with respect to the security level they provide and the standardisation efforts still required. For example, Option 1 would give the user access to the repository's policy. Option 2 requires agreement on the token format used.

Technical specifications implementing both policy-based and token-based solutions also exist. The OAuth protocol [HL10] specifies how an authorisation server can grant access to a resource by issuing an access token. The XACML Administration and Delegation Profile [RL14][1] allows policy-based delegation.

In Subsection 8.2.3, we describe an extension of our architecture. The properties included in the delegation models mentioned above are useful for this purpose (although not all of them are strictly necessary), but not sufficient: for our solution, we also neededa more generic way to express the permissions to be delegated. Regarding the delegation mechanism, we will rely on a delegation service (Option 5).

[1] This profile is not yet an official OASIS standard.

3.4 Model-Driven Security for Workflows

Several researchers have pursued the approach of annotating security requirements to graphical workflow models, with the ultimate goal of transforming them automatically to the enforcement level.

Wolter and Schaad [WS07] have proposed an extension for BPMN to model authorisation constraints. The extensions comprise role specifications for manual tasks, grouping of tasks, and artifacts for cardinality-based authorisation constraints. Wolter et al. [WSM07] build upon this modelling extension, presenting an approach that creates XACML policies for annotated BPMN diagrams. For each manual task, it creates corresponding XACML rules. Cardinality constraints are transformed into XACML conditions that use a custom function called `check:history`. However, the authors do not address how a BPMS can provide for the evaluation of this function. In [WMS+09], Wolter et al. describe the transformation framework in more detail.

Menzel et al. [MTM09] analyse some security requirements of workflows. They define BPMN artifacts to express organisational trust and what they call *security groups*, i. e. groups of tasks for which workflow designers can specify security properties. They define measures for the value of an asset and for trust ratings, which are used as properties of the newly defined artifacts, e. g. to describe how severe a breach of confidentiality during the execution of tasks would be. The transformation starts from a domain-independent security model and uses so-called security patterns which are based on expert knowledge of (domain-specific) security strategies.

Rodríguez et al. [RFMP07] have also proposed a BPMN extension for security requirements. They specify graphical notations and define the BPMN elements to which they may be attached. A model-driven approach is suggested to enforce the security requirements.

All these works deal with generating policies from BPMN annotations. They differ regarding the semantics of the annotations and how they are transformed into policies. The approaches of Wolter and Schaad as well as Rodríguez can be adapted to generate the policies needed for workflows in our architecture from BPMN annotations. The work of Menzel et al. addresses a very specific problem and is of little relevance to this thesis.

3.4.1 The SECTET Model

Hafner and Breu [HB08] present a model-driven approach for the systematic design and implementation of security-critical applications in service-oriented architectures. Compared to the approaches mentioned earlier in Section 3.4, their approach is more comprehensive and does not focus on annotations to graphical BPMN models. The aim of model-driven security engineering is to close the semantic gap between the security concepts used in the various development phases. This also reduces the amount of work needed to adapt an application to new business or security requirements. To this end, the authors have developed the SECTET framework for model-driven configuration and management of security infrastructures. SECTET defines a modelling component, a reference architecture based on web services, and a transformation component that translates model information into configuration code for the components of the target architecture. Using SECTET's domain-specific language (DSL), so-called

global workflows spanning multiple organisations as well as local workflows are defined. Together, these comprise the workflow view. They are accompanied by a so-called interface view, which comprises an interface model, a role model and a document model. Security policies are defined and relate to elements of both views. SECTET's reference architecture "specifies a component infrastructure based on Web services technology and specifications". It is based on the XACML interaction model, extended by other components. The resulting architecture is used as the target of model transformations. In summary, workflow models are transformed into WS-BPEL and security configuration results in corresponding XACML 2.0 policies.

SECTET focusses on the model-driven approach and the transformation into the target architecture. The target architecture specified is a means to achieve this goal. Even with a model-driven approach, it is still necessary to define a sensible execution architecture. This is a challenging task, as interoperability with existing specifications should be ensured. This thesis focusses on providing an execution architecture and leverages existing technologies. The resulting architecture can be seen as a target architecture in SECTET terminology. Accordingly, both approaches can complement each other. SECTET models the entire system and generates code for the target architecture. By contrast, our approach is to respect the independence of parties. We have developed the architecture of a workflow management system which interacts with other autonomous parties in a service-oriented architecture. Instead of using model transformation for all security problems discovered, we rely on autonomous subsystems that already provide the necessary security functionality. For example, SECTET uses an *Authentication and Role Mapping Unit* and specifies its own role model. We in turn rely on authentication information provided by a federated identity management infrastructure.

3.5 Bertino et al.

Bertino et al. [BCP06, BMPS09] describe the RBAC-WS-BPEL authorisation model. The model consists of a role hierarchy, permissions to execute activities, assignment of permissions to roles, and constraints expressed as relations that must hold for the users executing two given activities. The permissions are coded as XACML `Policy` elements. The roles are also expressed as `PolicySet` elements. To represent the role hierarchy, the `PolicySet` elements for roles refer to those for the comprised elements. The `PolicySet` elements for roles also refer to `PolicySet` elements encoding the role-permission assignment relation. For the constraints, a special language called *Business Process Constraint Language* (BPCL) is introduced. The authors also define a simple architecture that includes a policy enforcement point for user requests and a policy decision point that stores both kinds of policies. BPCL supports arbitrary predicates as constraints. However, evaluation of the predicates is not addressed.

Paci et al. [PFB09] extend this architecture in order to assign roles based on identity attributes in a privacy-friendly way. They introduce a component called *identity manager* that stores cryptographic certificates attesting the user's attributes. Based on a cryptographic protocol performed by the enforcement service and a client component running on behalf of the user, the enforcement service can determine whether the user has the attribute values

required to perform a certain task. This approach does not address privacy with respect to identifiers when enforcing constraints.

Our approach for policies is similar to RBAC-WS-BPEL. Regarding constraints, we use a simpler approach and focus more on the architecture used for their evaluation. The approach of Paci et al. can complement any architecture involving access control based on users' attributes, including ours. However, we use attributes not only for access control, but also for other, application-specific purposes.

3.6 Non-Functional Aspects in Workflows

Charfi [Cha07] has addressed the modularity of workflow specifications with respect to cross-cutting concerns. This is important for the integration of non-functional aspects, especially security aspects, because many of them constitute cross-cutting concerns. To this end, Charfi adapts the approach of aspect-oriented programming (AOP). He proposes a concern-based decomposition of workflow specifications and presents AO4BPEL, an aspect-oriented workflow language.

In detail, aspect-oriented languages have to define a model for join points, pointcuts and advices. Join points are points in the execution of a workflow that can be altered somehow by constructs of the aspect-oriented language. A pointcut is a language construct used to select related pointcuts. Finally, an advice, expressed in an advice language, defines the crosscutting functionality "that needs to be executed at the set of join points that are captured by a pointcut." AO4BPEL defines two kinds of join points: the execution of BPEL activities, and certain internal points during the interpretation of messaging activities. The pointcut language is based on XPath expressions. It can select activities directly or based on variables or partner links. One example is the selection of all activities sending messages through a given partner link. The advice language in AO4BPEL is WS-BPEL, i. e. advices are WS-BPEL activities. The input and output variables of the join point are made accessible to the advice code. The same holds for information about the join point and for messages processed at the joint point.

Charfi presents different applications of AO4BPEL. In particular, he develops a framework for the integration of middleware support into WS-BPEL workflows. Regarding non-functional requirements, he distinguishes messaging-level requirements and workflow-level requirements. The former are associated with messaging activities which send or receive SOAP messages. The latter, by contrast, are associated with structured activities and cannot be directly associated with individual SOAP messages. With respect to security, Charfi identifies a number of message-level security requirements that can be enforced using WS-Security. As an example of a security-related workflow-level requirement, he mentions the more efficient implementation of an exchange of several messages with one communication partner. To implement the aspects, he provides middleware services which are integrated into the workflow through calls in the advice code. This allows the integration of functionality which cannot be implemented in WS-BPEL.

Compared with AO4BPEL, our approach focusses on an architecture for secure workflow execution based on existing BPEL engines. Our PEP-HT and

PEP-WS can be compared to the middleware services in AO4BPEL. Nevertheless, we use a hard-coded BPEL transformation because we do not require the additional flexibility that AO4BPEL provides (and thus the additional complexity it requires). Similar to Charfi, we also have to deal with workflow-level requirements, in particular authorisation constraints. While AO4BPEL facilitates the integration of non-functional aspects, it does not reduce the complexity of actually implementing solutions to specific workflow-level requirements, which is one of the main topics of this thesis.

3.7 Support for manual activities in workflows

WS-BPEL itself does not address the integration of manual activities performed by humans. This issue was the subject of independent specifications. BPEL4People [AA+07b] specifies the integration of human activities into BPEL workflows. It introduces a new kind of activity called *people activity*. WS-HumanTask [AA+07a] defines the behaviour of applications presenting tasks to users. It provides a state model for tasks and defines so-called *generic human roles* representing actions that humans can perform on tasks. *People queries* allow dynamic allocation of users to activities. However, the specification does not define the format of these queries.

Both specifications do not address any security requirements. People queries provide a way to integrate authorisation mechanisms for human tasks, but we also need to support SSO in the worklist handler and use the attributes provided through SSO. Therefore, we use a custom simpler approach to represent and handle human tasks.

3.8 Identity management and workflows

In this section, we will address two other approaches which combine workflows with identity management.

3.8.1 Hummer et al.

Hummer et al. [HGS+11] present an approach for identity and access management in SOA. This approach is based on a new language for specifying an organisational RBAC model. In this language, one can specify subjects, a role hierarchy, resources and operations. Permissions are defined using roles, resources and operations. In addition, one can specify so-called contexts and define permissions that are only valid in a certain context. The language also supports separation of duty, but only in the static form. The underlying scenario of the approach is an inter-organisational service-oriented architecture where each organisation provides services and defines its own RBAC model. Workflows, which are expressed in WS-BPEL, invoke services from different organisations using the credentials of users with sufficient permissions. In [HGS+11], the scopes in the workflows are annotated with contexts and roles. Contexts represent a special state of the workflow, such as an emergency condition. At runtime, workflows get user credentials (name and password) from a special service, the *credential provider*. The user enters their credentials at the machine running the credential provider. Using these credentials, the workflow

retrieves a SAML assertion with the user's identity, their current role, and the workflow context from the identity provider (it is not mentioned how the identity provider knows about the workflow context, but apparently it is provided by the workflow). If the user is registered with another organisation's identity provider, the request is forwarded to it. When the workflow has retrieved the SAML assertion, it uses it in all subsequent calls to web services that require the same role and context. To achieve all this, Hummer et al. propose an automatic workflow transformation. In short, calls to the credential provider and the identity provider are inserted at the beginning of scopes, and the SAML assertion received as a reply is used as a SOAP header. The service invocations are intercepted by a policy enforcement point (PEP). This PEP checks the validity of the SAML assertion and then requests authorisation from a policy decision point (PDP). The PDP decides the request based on the RBAC model of the respective organisation, i. e. the one providing the service.

This approach has several shortcomings. (1) According to this approach, workflows handle unencrypted user passwords in order to retrieve SAML assertions from an identity provider. The authors refer to this process as single sign-on (SSO), but it is not, as in line with established terminology, login credentials are (only) handled by a trusted identity provider when performing SSO [Cha09]. (2) The approach uses an ad-hoc solution to acquire users' identity information, in particular, SAML assertions attesting their identity. By contrast, the approach presented in this thesis is more realistic, as it adheres to widely-used protocols specified for this purpose. (3) The approach only addresses the static version of authorisation constraints such as separation of duty. (4) The evaluation only addresses performance issues, and yields rather obvious results. It does not address the security properties of the resulting system.

3.8.2 Identity Management and Workflows

Görig [Gör09] extends a BPEL engine so that workflows can handle identity tokens. His implementation is based on WS-Security [LK06b], but abstracts from the actual token type used. He proposes two different mechanisms and respective BPEL extensions to this end. First, an attribute for incoming message activities (such as `<receive>`) specifies a variable into which the user context represented by the identity token attached to the message is saved. The workflow can use this variable in outgoing message activities. A token is then attached to the outgoing message when the activity is executed. Second, Görig introduces special functions to access parts of a token, such as the username or the value of an attribute. The token is specified by referring to the activity that has retrieved it. Görig also proposes a declarative mechanism to specify tokens used in outgoing messages outside of the BPEL code. The specification mentions two activities – the activity that has retrieved the token, and the activity that is to use it for an outgoing message.

Compared to Hummer et al., this solution only encompasses a technical mechanism for dealing with tokens in a BPEL engine, not an entirely new architecture. The solutions used in our architecture for accessing attributes and for specifying the identity to be used in outgoing web service calls are conceptually similar to those described by Görig.

3.9 Solutions available on the market

As SOA is more and more widely deployed, commercial and open-source systems have picked up security issues as well. The question now is which security functionality is available in commercial and open-source solutions. These solutions often support different kinds of web-service applications, the BPEL engine being only one module of a more comprehensive solution. We have thus looked at security functionality for web services in general, as well as for workflows in particular. Features we are interested in are transport security, access control, auditing, and how the respective functionality can be configured.

We have investigated the following products:

- Apache ODE (*Orchestration Director Engine*)[2] is a BPEL engine originally developed by Intalio and maintained as a project of the Apache Software Foundation. ODE does not provide any dedicated security functionality, but it is possible to integrate Apache Rampart for WS-Security support.

- IBM Business Process Manager[3] provides role-based access control for people interacting with a workflow through people activities. It also allows for access control for administrative tasks, such as inspecting the state of running workflow instances.

- Oracle BPEL Process Manager[4] support policies for web-service interfaces. These policies can specify message protection, authentication (based on SAML), and authorisation.

- JBoss jBPM[5]'s documentation mentioned that the "[s]ecurity features of jBPM are still in alpha stage" in version 3, but no longer does for the current version. Apparently, security features are not actively developed at the moment.

The conclusion is that solutions available on the market sometimes do offer support for existing SOA security protocols. The most advanced support is that in Oracle BPEL Process Manager. However, this support is not specially adapted to workflows. In particular, configuration cannot refer to workflow context. Consequently, there is no support in existing BPEL engines for more advanced security requirements. Therefore, we only require a bare BPEL engine and do not rely on proprietary security functionality.

3.10 Conclusions

We have reviewed related work from different areas and compared it with each other and our own work presented in this thesis. Some of this work contains interesting ideas. However, there are various shortcomings: Some approaches are based on wrong technical assumptions, and some are not directly applicable to either workflows, SOA, or both. To summarise, there is still no coherent

[2] `http://ode.apache.org/`
[3] Version 7.5; see `http://www-01.ibm.com/software/integration/business-process-manager/`
[4] `http://www.oracle.com/technetwork/middleware/bpel/overview/index.html`
[5] `http://www.jboss.org/jbpm/`

and integrated solution for secure workflows in SOA. Some of the related work addressed here will also be considered in the assessment of the state of the art (Subsection 5.1.4).

Chapter 4

Use Cases

The research presented in this thesis was conducted in the course of the TAS^3 (*Trusted Architecture for Securely Shared Services*) project, an integrated research project funded by the European Union's Seventh Framework Programme. TAS^3's goal has been to develop a service-oriented architecture tailored for the processing of personal data. A focus has been on decentralised data storage and processing, the dynamic composition of services used in workflows, and finally increased control opportunities for end users.

To this end, the TAS^3 project has focussed on the e-health and e-employability domains. These domains are well-suited to providing representative applications for the solutions developed by TAS^3. In both domains, many different parties are involved. Moreover, applications in both domains handle sensitive personal information protected by law.

This section introduces use cases from both domains. They are based on scenarios from the application partners of the TAS^3 consortium as part of the *Demonstration* work package (WP9) [CWK09].

4.1 Employability Scenario 1: Accreditation of Prior Learning (APL)

Accreditation of Prior Learning (APL) is a procedure for the recognition in higher education of qualifications acquired on the job, so that these qualifications can count towards a degree (possibly after some additional training). In the Netherlands, APL is regulated by collective labour agreements and provided by specialised companies, and paid for by the employer. The APL scenario has been taken from the business of the Dutch employability company Kenteq[1].

APL is representative for use cases in this and other domains: First, it processes personally identifiable information (PII), in this case information related to the career history and job qualifications of a user. Further, it involves experts at all stages. These experts need access to the PII of a specific candidate during workflow execution. We use the term *e-portfolio* for a repository that documents the user's work experience, education, professional career and credentials. APL uses web-service calls to integrate external applications such as e-portfolio storage.

[1] The explanation of APL in this thesis is based on Section 3.1 of [TAS09a].

Several human actors participate in the APL procedure:

- The *candidate* is a user who applies for APL and does not belong to the Kenteq organization.
- The *organiser* is a Kenteq employee charged with administrative tasks.
- The *coach* helps the candidate complete their e-portfolio (i.e. a collection of formal qualifications and work experience not yet formally captured) and checks available evidence (such as diplomas).
- The *assessor* executes the actual APL procedure. To do so, they take formal and informal qualifications into account. Based on this, the assessor validates their equivalence with certain other formal qualifications, possibly after having requested that the candidate complete additional training.
- The *quality controller* ensures that the procedure meets the quality standards of the APL code, and approves the quality of the report.

The scenario was modelled with support from Intalio using a top-down approach, the so-called *Process Modelling Framework* (PMF) [GD07]. This top-down modelling approach is based on various layers of workflow diagrams, from one high-level diagram covering the overall flow (the "core" workflow) via *phase*-level diagrams down to detailed *scenario*-level diagrams. This resulted in a workflow with the following parts:

1. The *Commence APL* phase contains administrative tasks needed to initiate the APL procedure.
2. In the *PCP Generation* phase, the candidate and Kenteq employees jointly create and validate a personal competencies profile (PCP). The candidate can agree to store the PCP with an e-portfolio provider available as a web service.
3. The *Reporting* phase assesses the abilities of the candidate and creates a formal report.
4. The *Procurement* phase encompasses administrative activities like creating an invoice or storing the final report in the e-portfolio.

Appendix A contains the resulting BPMN diagrams.

4.2 Employability Scenario 2: Student Work Placement

Another scenario from TAS[3] is the student work-placement scenario. The motivation for this scenario[2] is the importance of finding appropriate employment opportunities for university graduates in the UK and elsewhere. This has lead to increasing activity in the area of student work placement, which has yielded a complex network of funding bodies and placement schemes. Managing such programmes is becoming even more difficult because many universities rely on

[2]The presentation here is based on Chapter 4 of [CWK09].

the services of specialised placement providers. Consequently, the respective requirements need to be captured so that the process can be performed automatically as much as possible. A characteristic of this scenario is that sensitive and personal data is exchanged between several parties involved: The student, their university, the placement provider, and companies offering employment. This means that flexible trust and privacy settings are important. From a technical standpoint, this scenario involves the integration of services with existing data sources. The central processes are not automated and involve manual actions by office workers.

The scenario comprises the following steps:

- The student registers with a placement provider. During registration, their affiliation with a specific university is confirmed, and status information (e.g. the degree programme the student is taking) is transferred to the placement provider.
- The placement provider uses the status information to determine which placement programmes the student is eligible for.
- The student selects a programme they are interested in.
- The student provides information specific to the programme by uploading a file in a standard format or allowing direct access to an e-portfolio.
- They set a policy for this data and specify their trust requirements with respect to services involved in the placement procedure.
- The student selects a matching service from a list of services meeting their trust requirements.
- The placement provider releases the student's data to the chosen service, which then sends back a list of matching jobs.
- The student can apply for any of these jobs, but this is outside the scope of the placement procedure.

Because this scenario was only elaborated in a late stage of the TAS^3 project, it was not used for the requirements analysis. However, the concepts developed based on the APL scenario are equally applicable to the work placement scenario.

4.3 Travel-Booking Example

The use cases in TAS^3 have been derived from real-world scenarios. Because the procedures in place are heavily influenced by already mature technology, they are, however, not fit for demonstrating advanced identity-management features. For this reason and for ease of understanding, we needed a more intuitive and straightforward use case.

As such, we chose a corporate travel-booking application encompassing services for booking a flight, a hotel and a rental car. A clerk is involved, to check whether the intended booking confirms to corporate guidelines for travel expenses. The corresponding workflow consists of the following activities:

- *Initiate Booking*: A web-service call starts the workflow. It contains the destination, travel dates, and the reason for the trip. Only regular employees may start the workflow.
- *Authorise Travel*: A manager has to authorise the trip based on the destination, travel dates, reason of the trip, and name of the traveller. Note that a manager may not approve their own trip.
- *Choose Airline*: The traveller who started the workflow can choose one of the airlines available for the trip in question.
- *Book Flight*: The flight is booked by means of a web-service call to the travel agency. This call is made on behalf of the traveller.

We can think of additional functionality that would enrich this scenario:

- The travel-booking workflow could offer custom functionality based on user attributes. For example, it might use the traveller's professional status to determine the booking class (business or economy) they are eligible for.
- Other services based on the traveller's identity could be involved: As part of the booking workflow, itinerary data could be entered directly into the traveller's calendar, or a service could authorise payment if the traveller has given consent via a more secure channel.

Chapter 5

Requirements Analysis

In this chapter, we analyse the requirements of the architecture envisioned, in two steps. As the first step (Section 5.1), we consider the security of workflows in service-oriented architectures with a broad scope. We systematically derive security requirements based on a classification of security goals and on different workflow aspects. We assess the state of the art with respect to these requirements[1], and formulate high-level design goals for a secure workflow management system. In the second step, we address the integration of federated identity management (FIM) into a workflow management system (Section 5.2). To this end, we first set design requirements that a workflow management system should follow. We then consider how individual pieces of FIM functionality can be implemented in such a system[2].

5.1 Overall Security Requirements

In this section, we determine security requirements of workflows (and workflow management systems) in service-oriented architectures. First, we set out the methodology used (Subsection 5.1.1). We then perform the requirements analysis according to this methodology (Subsection 5.1.2) and assess the resulting requirements with respect to the state of the art (Subsection 5.1.4). Finally, we define and explain high-level design goals for the implementation of a secure workflow management system (Subsection 5.1.5).

5.1.1 Methodology

We have structured our analysis of security requirements specific to workflows in service-oriented architectures along two dimensions. The first dimension takes the different perspectives on workflows (Subsection 2.3.3), i. e. the functional, the behavioural, the organisational, the informational and the operational aspects, into account. Security has also been described as a workflow aspect by some researchers. However, we consider it separately, as we want to analyse its relationship to the perspectives listed above. For the second dimension, we rely on a widely used taxonomy for security functionality. While

[1] This part of the chapter was partly published in [MMvSB10].
[2] This part of the chapter was published in [MB14].

there are some differences in terminology between various textbooks and standards (cf. Subsection 2.4.1), they agree in fundamental points. Here, we use the multiple security frameworks listed in ITU-T Recommendation X.810 (cf. Subsection 2.4.1.3). In the following, the term *security facets* is used for the functionality described by the frameworks, i. e. authentication, access-control, non-repudiation, confidentiality, integrity, security audit and alarms, and key management.

All security facets are essential to yield a secure workflow execution. In the following, we will focus on workflow-specific security requirements, but naturally, there are requirements that are not specific to workflows but need to be fulfilled as well. For instance, ensuring the confidentiality of web-service invocations is no different for workflows and other applications in service-oriented architectures.

5.1.2 Requirements Analysis

The structure of this section is based on the different security facets. For each of these, its relationship to the different workflow aspects is explored. We illustrate the requirements with examples from the APL use case.

5.1.2.1 Authentication

Authentication means verifying that something is genuine or original. In the workflow context, this refers to the different entities involved in a workflow and to their properties.

The functional and behavioural perspectives describe the elements and structure of the workflow, which have to reflect the business objectives of the service provider running the workflow (i. e. Kenteq). This means the workflow model has to be created by or on behalf of the service provider and its genuineness has to be ensured.

Example 1 (Genuine Workflow Models)*: Kenteq's APL workflow contains the actions that need to be performed before a certificate can be issued, and the conditions that must be fulfilled. This procedure must be strictly adhered to in order to accomplish Kenteq's business objectives.*

Based on this, we formulate the following requirement:

- **R1**: Workflow models deployed on and executed by the system must be authentic, i. e. originate from the company running the workflow.

The informational perspective concerns data used in the workflow. This data can serve two purposes: First, it can influence the execution of the workflow itself through decision gateways. In this case, we have a connection to the behavioural perspective. Second, it can serve as input for operations in the workflow. In both cases, authentic *input* is necessary to get authentic *output*. However, authentic input is not sufficient: to ensure authentic output, the transformation has to be trusted.

Example 2 (Authentic Data)*: In APL, a so-called personal competency profile (PCP) is created that reflects the competencies of the candidate. It can be based on prior certificates (such as diplomas). This information is used*

to decide whether additional tests are necessary, and is the basis for the final APL certificate. Diplomas are issued by third parties (such as universities) and retrieved from the candidate's e-portfolio, which is maintained by another third party. Kenteq must be able to rely on the information provided by these third parties.

This results in the following requirements:

- **R2**: The system must check whether data relevant to the workflow comes from a trusted source.
- **R3**: It must be possible to specify which data sources or services transforming data are trusted.

The operational perspective concerns the services involved. The system must know which service a workflow is talking to, otherwise it cannot determine whether it is trusted, for instance. This also holds vice versa: Services must know which service provider has invoked an operation, and for which user.

Example 3 (Authentication for Web-Service Calls)*: APL uses data from an e-portfolio. Kenteq trusts certain e-portfolio repositories to provide authentic data. Kenteq also stores the final report in the candidate's e-portfolio. The repository needs to know that Kenteq initiated this operation on behalf of the candidate. The interaction with the repository can consist of several steps, like creating and later amending the e-portfolio.*

From this example, we can derive the following requirements:

- **R4**: The system must be able to authenticate services invoked by the workflow.
- **R5**: Services must be able to authenticate the service provider running a workflow that invokes their operations.
- **R6**: Workflows must be able to call services on behalf of users. Services must be able to authenticate the user on whose behalf a workflow invokes the service's operations.
- **R7**: Service providers must be able to link messages referring to the same user, throughout one or more workflow instances.

Finally, the actors involved in the workflow are important (organisational perspective) – they influence the execution workflow. Thus, only certain users may perform tasks. In addition, certain tasks must be performed by the same person, or may not be performed by the same person (as detailed below). Not all participants are directly known to the organisation running the workflow.

Example 4 (Authentication of Users)*: Kenteq has specific employees that may act as assessors. Assessors may not act as coaches in the same APL case. The candidates have likely did not have previous contact with Kenteq before the APL procedure, but have accounts with their current employer's system that they would like to use for Kenteq as well.*

From this, the following requirements result:

- **R8**: The system must be able to reliably determine the properties of actors.
- **R9**: The system must be able to determine whether a user is the same as one who has previously accessed the system.
- **R10**: Users should be able to use existing accounts for authentication.

5.1.2.2 Access Control

The goal of access control is mainly to prevent unauthorised use, disclosure and modification. In connection with workflows, this concerns different kinds of entities which relate to the different workflow perspectives. These entities include the start of workflow instances and the execution of workflow activities (both relating to the functional perspective), decisions that affect the execution flow (relating to the behavioural perspective), and accessing data related to the workflow (relating to the informational perspective). Access control for these entities depends on the user seeking access, thus creating a connection to the organisational perspective.

Workflow Start

Example 5 (Workflow Start)*: A user gets an APL voucher from their employer. Their permission to use APL is connected to their account. They go to the Kenteq website and can start APL. The system assigns this APL instance to them.*

- **R11**: A user must be able to start a workflow instance if and only if they have the necessary credentials.

Actually, starting a workflow is merely a special case of performing a task in it. Thus, the requirements in the following subsection also apply.

Workflow Activities The control flow of workflows is driven by the execution of activities. The WfMS performs some activities autonomously, but most require a user to perform them. In the latter case, access control mechanisms determine who may perform which activities. From the following examples, we can derive various requirements that the access control mechanisms must meet.

Example 6 (Explicit Manual Assignment)*: APL requires the coach and assessor to be familiar with the field the candidate works in. This decision is not taken based on a fixed security policy. Instead, in the* Allocate Resources *task, the organiser chooses the users to perform these roles based on business-related criteria. This choice is valid for a particular APL case. However, each coach/assessor assigned manually must still fulfill the general criteria for these positions.*

- **R12**: It must be possible to group tasks in a workflow model with common responsibility.

- **R13**: Assignment of responsibility must be possible based on an explicit decision by a human actor or by some workflow logic.
- **R14**: Assignment must be possible on the workflow instance level, i. e. different assignments for each instance.
- **R15**: The system must control whether someone manually assigned responsibility for certain tasks meets the criteria these tasks demand, i.e. possesses the necessary roles or attribute values.
- **R16**: Explicit assignment of responsibility (R13 has to be subject to the same restrictions as (other) tasks (in particular R13 and R17).

In contrast to this example, the next example concerns the allocation of tasks when no manual assignments have been made.

Example 7 (Automatic Allocation of Tasks)*: The APL procedure requires an organiser to perform some purely administrative tasks. Any available clerk can perform them – it does not matter who.*

This leads to the following requirement:

- **R17**: When no user has been explicitly assigned to a task, the system should allow any available user to perform it, as long as they are eligible.

The following examples introduce requirements to limit which tasks users can carry out based on the relationship between tasks. Some of these requirements are well-known from the literature.

Example 8 (Binding of Duty)*: The duties of actors involved in APL are often interrelated. For instance, the assessor has to write a report based on their interview with the candidate. This means that the same person must perform these two tasks (the interview and report). The workflow is started on behalf of the candidate, who has to perform several tasks in it.*

This example leads to a requirement known in the literature as binding of duty (cf. Subsection 2.4.3.2):

- **R18**: The system must allow the specification of a single actor to perform all tasks in a group of related tasks.
- **R19**: It must be possible to specify that the user who started a workflow instance must also perform other tasks in this instance.

Example 9 (Reassignment of Duty)*: The APL for candidate John has been running for some time. His coach, Bob, goes on vacation. Before doing so, he agrees with Jane that she will take over coaching John, so that John's APL is not delayed. Like Bob, Jane is a qualified coach for John's employment domain.*

This example shows that strict binding of duty is sometimes unsuitable, even in non-emergency cases. Accordingly, a controlled mechanism to override it is necessary. The requirements arising from this example are as follows:

- **R20**: The system must allow actors to reassign their involvement in a workflow, including their responsibilities and privileges to perform tasks and access related data.
- **R21**: The user to whom the assignment is transferred must fulfil the applicable constraints, just like the original assignee (as specified in R15 and R17).

The following example shows a possible conflict of interest in the APL usecase. Such conflicts are usually remedied by introducing separation of duty (cf. Subsection 2.4.3.2, as well).

Example 10 (Conflicts of Interest)*: In any APL case, the coach helps the candidate complete their e-portfolio, and the assessor examines it. There is a conflict of interest between these tasks, as the assessor is implicitly evaluating the quality of the coaching as well. Further, an assessor's request to revise the e-portfolio will incur additional work for the coach.*

The resulting requirement is as follows:

- **R22**: The system must enable the specification of tasks that are in conflict. It must not assign the same user to conflicting tasks in the same workflow instance.

Data In the following example, we examine how data, especially personal data, is used in the APL workflow.

Example 11 (Use of Externally Stored Data)*: A coach and an assessor are in charge of an APL candidate. To fulfil their duty, they need access to the candidate's personal data, such as a diploma stored in their e-portfolio. The candidate must agree to this use when they start APL. The assessor may only access the e-portfolio once the coach has approved this. Kenteq might also outsource steps of the APL procedure to external service providers who need access to the data as well.*

This example results in the following permissions:

- **R23**: The system must grant human actors in a workflow the permissions they need to perform their tasks.
- **R24**: The system must allow workflow designers to specify which kind of data is needed for which tasks.
- **R25**: The system must let users permit the workflow and the actors involved in it to access their personal data.
- **R26**: When external services are involved in the workflow, the system must be able to grant them access as well.
- **R27**: The system must not allow access (according to R23 and R26) to data not relevant to the current workflow instance.
- **R28**: The system must be able to restrict access to the time when particular tasks are being executing.

Web Services Workflows invoke web services to achieve their functionality. Personal data is disclosed to the services, and these services influence the outcome of the workflow. Consequently, controlling the choice of services is crucial.

Example 12 (Web Services)*: A new APL candidate does not yet have an e-portfolio, so the workflow needs to access a service provider to create one. Several providers are available, and the candidate wants to select one they trust. As the candidate often has to make these types of choices, they want to reuse their usual settings. At the end of the APL procedure, the* Upload PCP *task stores the PCP in the e-portfolio.*

This gives rise to the following requirements:

- **R29**: The workflow must respect user settings when selecting service providers. If specified, the user must be able to confirm the service selection.
- **R30**: Users must be able to choose service providers based on certain trust indicators.
- **R31**: The system must ensure that only service providers trusted by the organisation running the workflow are used. This is a generalisation of R3 that is applicable as well e. g. to services data is only sent to, but not received from.

5.1.2.3 Non-repudiation

Non-repudiation as a security goal means that the originator of some action or information cannot deny it.

Example 13 (Reliance on results from business partners)*: When Kenteq outsources some part of the APL procedure, it must be able to rely on the result. In particular, the company wants to have legal remedies at its disposal for the case that incorrect results occur due to negligence on the part of business partners. This is only possible if Kenteq can prove that the result originates from a particular business partner. Conversely, Kenteq stores the complete APL certificate in an e-portfolio. Because other organisations (such as future employers) rely on the content of the e-portfolio, Kenteq is responsible for the accuracy of the APL certificate. Thus, proof is needed that the certificate was issued by Kenteq.*

This results in the following requirements:

- **R32** Messages from external service providers to the workflow should be attributable to them, with minimal risk of repudiation.
- **R33** Messages from the workflow to external service providers should be attributable to it, with minimal risk of repudiation.

5.1.2.4 Confidentiality

Example 14 (Sensitive data in web-service calls)*: When the APL workflow calls external services, sensitive personal data is transmitted. The candidate wants their data to be protected from disclosure to others. Both Kenteq and the external service provider have a legal obligation to provide such protection. The same holds when personal data is shown to the coach or the assessor.*

This results in the following additional requirements:

- **R34**: Messages from the workflow to external service providers must be protected against unintended disclosure to third parties.
- **R35**: Messages from external service providers to the workflow must be protected against unintended disclosure to third parties.
- **R36**: Data shown to workflow participants must be protected against unintended disclosure to third parties.

Example 15 (Unrelated workflow instances)*: A candidate performs APL at Kenteq because it was offered to them by their employer. Later, they use Kenteq's services to look for a new job. The user does not want the two cases to be linked without their consent.*

Based on this example, we can formulate the following requirement:

- **R37**: It should not be possible to link user identities used in unrelated workflow instances.

5.1.2.5 Integrity

Integrity means that something is complete and uncorrupted, and can apply to different entities involved in a workflow. Integrity means protection against unauthorised modification, deletion, creation, insertion and replay [ITU95b]. When integrity is provided for the system the WfMS is running on, its data storage and its communication links, and the WfMS performs access control correctly (according to the requirements introduced above), no special workflow requirements remain. For this reason, we only state one very generic requirement:

- **R38** The system must ensure the integrity of workflow definitions, the data used in workflows, and the communication performed by workflows.

5.1.2.6 Security Audit and Alarms

With respect to workflows, security audits must capture all entitities involved in a workflow, as captured by the different workflow perspectives. Another important factor is who needs to have access to the audit information.

Example 16 (Audit)*: Candidate John goes through APL. His first profile is not accepted by the assessor, so he has to revise it. In the meantime, his first coach goes on vacation, so he is supported by another coach the second time. At the end, based on John's decision, a new e-portfolio is created and his certificate is stored in it.*

This example illustrates the following audit requirements:

- **R39**: The system must let authorised users audit which tasks were executed and which branches were taken in a workflow.
- **R40**: The system must let authorised users audit who performed each task in the workflow.
- **R41**: The system must let authorised users audit which services a workflow has invoked.
- **R42**: The system must let authorised users audit which data was processed or transferred in the workflow.

Example 17 (Information for End Users)*: John is interested in finding out how his data was used by Kenteq. He also has a legal right to access this information.*

This leads to the following requirement:

- **R43**: Users should be able to access audit information concerning workflows that have handled their personal data.

5.1.2.7 Key Management

Cryptographic keys are used mainly to ensure confidentiality and non-repudiation. Key exchanges are connected with the establishment of trust relationships. The first question here is which kinds of data and messages require encryption or cryptographic signatures. Following on from this, the second question is how trust relationships are established.

Example 18*: Trust Relationships A candidate should be able to access the workflow via their existing account with their employer.This means the candidate's employer must authenticate them and notify Kenteq of the authentication.*

During the workflow, web services are discovered and invoked. This invocation requires a trust relationship to ensure non-repudiation and confidentiality.

This example results in the following requirements:

- **R44**: It must be possible to configure trusted sources of authentication information.
- **R45**: The system must be able to retrieve the keys of service providers that it exchanges messages with.

5.1.3 Summary of Security Requirements

We have analysed the security requirements based on a categorisation of security functionality and workflow perspectives. The examples stem from a real-world application that handles sensitive data which demands an increased level of security.

On this basis, we deem these requirements representative for a broad range of applications. In a nutshell, they emphasise the use of workflow state for security decisions and the distributed, sometimes inter-organisational environments that modern workflows run in.

5.1.4 Assessment of the State of the Art

This chapter analyses whether existing technology solves the requirements pinpointed in the previous chapter or at least contributes to a solution. This assessment includes industry specifications (Chapter 2), proposals from the academic literature (Chapter 3) and available software[3].

The remainder of this chapter first establishes the technological baseline that the assessment is performed against. The assessment itself comes next. Its structure is based on the same security goals as the requirements analysis. However, it also takes the layers of a given system into account. Table 5.1 lists the section where each requirement is addressed.

5.1.4.1 Technological Baseline

This thesis does not try to build the entire architecture needed to fulfil the requirements from scratch. Rather, it focusses on workflows in a service-oriented architecture and builds on technologies already established in this field.

The respective technological baseline includes the core web-services platform as introduced in Section 2.2, and BPEL for service orchestration (Subsection 2.3.2). From this baseline we can perform a well-structured assessment as follows: First, we determine state of the art that can be combined with the technological baseline. Then, we analyse how to achieve this combination. The goal here is to provide a first step towards fulfilling the requirements. This reduces the amount of work needed, and facilitates interoperability of the architecture envisioned with existing technology.

5.1.4.2 Non-repudiation and confidentiality

R32, R33 (Subsection 5.1.2.3), R34, and R35 (Subsection 5.1.2.4) require non-repudiation and confidentiality for web-service messages exchanged between the workflow and external web services. Usually encryption is used to provide confidentiality, while cryptographic signatures guarantee that a message originates from a entity holding a certain signature key (non-repudiation). Both mechanisms rely on asymmetric encryption schemes like RSA [RSA78][4]. WS-Security [LK06b] makes it possible to encrypt and sign SOAP messages, meeting the above requirements.

WS-BPEL recommends secure transport using WS-Security. However, this is not mandatory. This also implies that the configuration is not standardised. Another open issue is key management, addressed below. AO4BPEL (Section 3.6) provides a possible way of integrating message handlers into BPEL workflows. However, it does not define how to implement the functionality itself, in particular how to integrate it with service discovery and key management.

R36 requires confidentiality for data shown to human workflow participants. When using a web interface to involve human users, HTTPS [Res00] provides confidentiality for the connection. In order to provide end-to-end confidentiality, integrity of the user's system must be ensured. However, this problem is not workflow-specific and outside the scope of this thesis.

[3]The content of this chapter was partly published in [MMvSB10].

[4]Encrpytion is also possible using only symmetric schemes, but this complicates key management.

Security goal	Area	Requirement	Assessment in
Authentication	Workflow Models	R1	Subsection 5.1.4.6
	Data	R2	Subsection 5.1.4.4
		R3	Subsection 5.1.4.4
	Web Services	R4	Subsection 5.1.4.8
		R5	Subsection 5.1.4.8
		R6	Subsection 5.1.4.8
		R7	Subsection 5.1.4.8
	Users	R8	Subsection 5.1.4.7
		R9	Subsection 5.1.4.7
		R10	Subsection 5.1.4.7
Access Control	Workflow Start	R11	Subsection 5.1.4.3
	Activities	R12	Subsection 5.1.4.3
	(including Allocation)	R13	Subsection 5.1.4.3
		R14	Subsection 5.1.4.3
		R15	Subsection 5.1.4.3
		R16	Subsection 5.1.4.3
		R17	Subsection 5.1.4.3
	Constraints	R18	Subsection 5.1.4.3
		R19	Subsection 5.1.4.3
		R20	Subsection 5.1.4.3
		R21	Subsection 5.1.4.3
		R22	Subsection 5.1.4.3
	Data	R23	Subsection 5.1.4.5
		R24	Subsection 5.1.4.5
		R25	Subsection 5.1.4.5
		R26	Subsection 5.1.4.5
		R27	Subsection 5.1.4.5
		R28	Subsection 5.1.4.5
	Web Services	R29	Subsection 5.1.4.4
		R30	Subsection 5.1.4.4
		R31	Subsection 5.1.4.4
Non-repudiation		R32	Subsection 5.1.4.2
		R33	Subsection 5.1.4.2
Confidentiality		R34	Subsection 5.1.4.2
		R35	Subsection 5.1.4.2
		R36	Subsection 5.1.4.2
		R37	Subsection 5.1.4.2
Integrity		R38	Subsection 5.1.4.10
Security Audit and Alarms		R39	Subsection 5.1.4.9
		R40	Subsection 5.1.4.9
		R41	Subsection 5.1.4.9
		R42	Subsection 5.1.4.9
		R43	Subsection 5.1.4.9
Key Management		R44	Subsection 5.1.4.6
		R45	Subsection 5.1.4.6

Table 5.1: Overview of requirements and their assessment

5.1.4.3 Access Control for Workflow Activities

We have stated a number of requirements relating to controlling who performs (manual) workflow activities. In Subsection 2.4.3.1, we introduced traditional, non-workflow-oriented access control models. These models do not contain any notion of a workflow or the activities one consists of. However, they can be applied to workflows in a straightforward way in order to perform access control for individual activities, by considering them to be resources. This way, R11 and R15 are fulfilled.

There are also several approaches for authorisation constraints based on the relationship between activities (Subsection 2.4.3.2). Support for binding-of-duty (R18, R19) and separation-of-duty (R22) constraints are described thoroughly in the literature. The approaches support instance-specific evaluation of constraints (R14). The work of Basin et al. [BBK11] provides a solution for constraints when loops exist in the workflow schema.

Most older access control models for workflows are based on RBAC. However, adaption to ABAC to fully support R15 is straighforward. For example, the architecture presented by Hummer et al. (Subsection 3.8.1) uses ABAC. We deem groups of activities (R12) important for coupling permissions on data (see Subsection 5.1.4.5 below) with assignments to these groups. There is no concept for activity groups and the assignment of users to them in existing models.

A reassignment mechanism (R20, R21) is also absent from existing models. Reassignment a type of delegation (Section 3.3) that transfers both privileges and responsibilities. We require bilateral agreement for a reassignment to take effect. In addition, reassignment is related to task groups. The reassignment mechanism overrides active binding-of-duty constraints. In this respect, it is similar to the overriding of constraints in W-RBAC (Subsection 3.2.4). Finally, the mechanism has to consider permissions on data.

In addition to the shortcomings of the theoretical models, existing specifications and systems have insufficient support for the integration of access control. BPEL4People and WS-HumanTask (Section 3.7) address the integration of manual activities into BPEL workflows. There is an extension mechanism that allows for dynamic allocation of users to activities, and could thus support the requirements mentioned so far. However, the format of these queries is not part of the specification. Mendling et al. [MPS08] have investigated how separation-of-duty constraints can be enforced using BPEL4People. They determine that this is possible in general, but not in all situations. One problematic case is that of two activites which can be performed concurrently.

One must be able to express authorisation requirements, including constraints. To this end, suitable policy languages are required. XACML (Subsection 2.4.3.3) can be easily profiled for workflow tasks. However, the RBAC profile does not address separation of duty. The policy languages proposed by Bertino et al. (Section 3.5) support role-based authorisation and constraints. However, they do not support activity groups and reassignment.

Finally, we require support for two different strategies for allocating users to activities: explicit allocation (R13) and automatic allocation (R17). It is possible to implement explicit allocation manually in a WS-BPEL workflow. The system still has to apply security mechanisms in order to fulfil R15 and R16. Automatic allocation to any eligible user is the default strategy in many

commercial systems such as Intalio BPMS 6. Further work is necessary to integrate allocation mechanisms and authorisation constraints.

5.1.4.4 Selection of Trusted Web Services

R30 and R31 require an infrastructure that can determine a trust score for web services, both from the perspective of users and the organisation running the workflow. Solutions to this problem are the subject of ongoing research. A generic policy-based solution must comprise a format to represent user preferences and tools to edit preferences. Moreover, the selection and confirmation by the user (R29) has to be integrated into the workflow execution. WS-BPEL allows workflows to select web services at runtime by explicitly defining appropriate code. We need a mechanism to easily integrate the necessary user interactions into existing workflows. Making sure that the services used are trusted by the organisation running the workflow also meets R2 and R3.

5.1.4.5 Access Control for Data Related to a Workflow

Roughly speaking, requirements R23 to R28 seem to suggest the need for a mechanism that grants access to data somehow related to the workflow. The permissions granted have to be in sync with the execution of the current workflow instance. Section 3.2 presents some models for access control in dynamic environments and workflows. We will now assess their suitability based on the above requirements.

TMAC assigns resources to cases. It does not take activities into account, so it cannot grant a permission just to a person performing a certain task. More specifically, two team members with the same role in the organisation but performing different tasks in the case are granted the same permissions. This means that TMAC does not fulfil R23 and R28. WAM fulfils the requirements at a basic level. However, it directly specifies the subjects (users or programs) that have to perform a task, not just the criteria for eligibility. WAM specifies the types of objects used and created in a task. WAM thus only grants permissions for the duration of an activity, not a whole sequence of activities. This contradicts R28. Authorisations in TBAC directly refer to subjects and objects. TBAC does not derive the actual subjects and objects from the workflow context, so it does not meet R23.

On a more technical level, the system must gain permissions on data related to the workflow and grant them to users needing access. In particular, R25 requires that users are able to explicitly grant permissions to the workflow. As stated in Section 3.3, this is an example of delegation, administered by the WfMS acting as an agent on the data owner's behalf. Both policy- and token-based solutions can be leveraged here.

5.1.4.6 Configuring Trust Relationships

R1, R44, and R45 all concern the configuration of trust relationships essential for the functionality of the WfMS: Who can deploy workflow models? Who can assert user properties, giving users access to activities and data? How can the WfMS be sure it is actually talking to a web service it considers trusted? From a technical perspective, this is achieved by making the respective credentials known to the system, i. e. for the administrative account allowed to deploy

workflow definitions, and public keys of identity providers. Authentic public keys of service providers are usually retrieved from a service registry.

5.1.4.7 Authentication of Users

R8 and R9 require that the system is able to determine users' properties and recognise users throughout the execution of the workflow. Users must be able to reuse their existing accounts to this end (R10).

To meet these requirements, the WfMS has to rely on information provided by trusted sources (R44, Subsection 5.1.4.6) in an FIM infrastructure. We have described FIM technologies in Subsection 2.4.5.3. SAML and ID-WSF support SSO, allowing users to reuse their accounts. They also provide users' attributes and identifiers to service providers. Support for SSO has to be integrated into the user interface of the WfMS, though. It is not present in current systems. In addition, it has to be coupled with the access control functionality (Subsection 5.1.4.3).

Recognising users throughout the workflow (R9) is only possible when the lifetime of the user's pseudonym allows this. R37 requires that workflow instances are prevented from combining information they possess about users. This combination would be easy if different workflow instances know a user by the same pseudonym. A solution would be different pseudonyms per workflow instance. However, this is not supported by the SAML framework, as it only provides different pseudonyms for different service providers.

5.1.4.8 Authentication and Identity Management for Web-Service Calls

R4 and R5 require that the service provider running the workflow and the one running services invoked by it are able to authenticate each other. Signing the messages exchanged is possible using SAML and WS-Security and meets these requirements. In addition, proper exchange of public keys is necessary (Subsection 5.1.4.6).

The ID-WSF protocol suite defines how to perform web-service calls with the identity of a user. This meets R6. However, integration into the WfMS is currently absent. ID-WSF also defines an identity-mapping service that supports different identifiers for each service provider. This mapping service allows service providers to recognise users independent of the workflow instance that has invoked the service. It accordingly meets R7.

5.1.4.9 Audit

The WfMC has specified an audit data format for workflows [WfMC98] and started to develop an XML-based successor [zM08]. Both formats focus on the execution only, i. e. the state of workflow instances and activities. They do not address data handling and the messages exchanged between workflow instances. Weske [Wes07] mentions a monitoring component that visualises the state of workflow instances but does not state what a visualisation of this kind should look like. The *Intalio Designer BPMS Console*[5] lists available workflow definitions and instances, and marks activities currently running. However,

[5] `http://www.intalio.com/bpms/designer`

it does not address data transfer and is targetted at administrators, not end users.

In summary, our assessment is as follows: Preliminary approaches for capturing audit information exist. They fulfil R39, but not R40, R41, and R42. The presentation of audit information is not suitable for end users and thus does not meet R43.

5.1.4.10 Overall Integrity

Overall system integrity (R38) is a very general requirement. A lot of other requirements contribute to it. The components used to build a WfMS also have to meet this requirement, and the environment has to meet the assumptions made when designing the system. As such, a final assessment regarding this requirement is not possible at this stage.

5.1.4.11 Summary

Service-oriented architectures can support applications with sensitive, personally identifiable information. We have assessed the requirements of such an application with respect to the state of the art. Existing SOAs already feature security mechanisms, but it is not clear how these satisfy the security needs of workflows. Some approaches work within specific problem areas only. All of this calls for a comprehensive analysis that can act as a baseline for designing mechanisms to secure workflows. We have evaluated related work from different sources and identified open issues. In particular, existing approaches address our requirements only partially. Moreover, most of them are not integrated into any WfMS. This assessment will be useful for designing security mechanisms for workflows that fit well into existing WfMSs and are embedded into an open security framework.

5.1.5 High-Level Design Goals

The functional requirements stated above are manifold and address different levels. For example, we have already hinted at aspects such as configuration. Our goal is to provide a workflow management system that fulfils these functional requirements. We can only achieve this with a focussed approach, otherwise we cannot expect any reusable insights or even to yield a useful architecture at all. Therefore, we will now formulate and explain design goals which will help us in the development and evaluation of our architecture[6]. They are derived from characteristics of service-oriented architectures, and their purpose is to aid in the implementation of the functional requirements.

Minimal Changes An existing workflow management system should require only a few changes. This has the advantage that a solution is easily adapted to other workflow management systems. Further, it might be possible to continue using existing tools, such as for modelling and testing.

[6]The design goals have been published in [MB11].

Standard Architectures The architecture should follow the XACML reference architecture and the WfMC workflow reference model where possible. This allows established technology to be used and will make the architecture easier to understand and implement.

Structure Preservation In our architecture, we support different kinds of security specifications, such as trust policies for service calls, attribute-based authorisation policies for human tasks, and history-based constraints which are dependencies between tasks. It is not obvious how to map these specifications from a higher layer, such as a task annotation in a workflow model, to the form used for runtime enforcement. To illustrate this, consider code generation in a compiler. In the resulting machine code, even without optimisation, much semantic information from the source code, such as variable names, is no longer available. With additional debug information, it may be possible to reconstruct this to a certain degree and use it to monitor program execution in a debugger, but not always. The situation with security specifications for workflows is analogous. For example, when the workflow compares the identifier of a user who tries to perform a human task against a variable, it may not be obvious that the purpose of this comparison is to enforce a separation-of-duty constraint. When a workflow fragment is inserted to perform a complex security-specific interaction, the information stating which rule has caused the insertion might not be available. We want to map security specifications to the execution level so that the structure of the individual rules is preserved. In the compiler example, this would mean that optimisation does not remove those variables. With separation of duty, the constraint should be expressed as a rule attached to a human task and evaluated when that task is performed. This makes security enforcement easier to understand and debug, and it eases auditing by making explicit *why* a security decision has been taken. Consequently, it must be clear which components of the architecture enforce which types of security specifications, and how the necessary information is captured and stored.

5.2 FIM Support in Workflows

In the previous section, we addressed a very broad range of security requirements for workflows. Now, we narrow the scope by considering the integration of workflows and a federated identity-management (FIM) infrastructure. First, this approach makes it possible to address some security features again from a different perspective. This is because FIM provides solutions for a wide range of security requirements. Second, this approach is more specific, as it takes concepts from existing FIM frameworks into account. Third, FIM has brought about features that can be seen as application functionality rather than security functionality. We will therefore address the orchestration of identity-based services using workflow technology.

5.2.1 Design Requirements

In Subsection 5.2.2, we will determine how different kinds of FIM functionality can be made available to workflows and integrated into a workflow management system. The overall goal of this analysis is the development of workflow

management system of this kind. The development should accomodate issues such as privacy and interoperability. In the following, we formulate design requirements that will serve as guidelines for further analysis.

We distinguish between privacy-related and architecture-related design requirements. The requirements belonging to the first group (D1 and D2) stem (1) from the laws of identity stating the desired privacy properties of FIM applications (Subsection 2.4.5.2). (2) The architecture-related requirements stem from best practices in the fields of workflow management and SOA, and (3) from considerations on how to facilitate implementation and system maintenance. Further, [Erl05] lists principles of service orientation leading to some of the requirements: Services abstract underlying logic (D3) and are autonomous and composable (D5). Contemporary SOA is based on open standards (D4).

D1 *Isolate workflow instances*: The laws of identity propose pseudonyms for users that are different for each service provider to prevent different organisations from combining their identity information about a user. The SAML framework provides a solution through pseudonymous `NameID`s that differ for each service provider. However, a workflow management system acting as a single service provider can host different independent workflows. This means that the workflow management system needs to prevent several workflow instances running in the same system from combining their identity information.

D2 *User consent*: Identity information should be released only with the unambiguosly expressed consent of the user. When the workflow management system provides identity information to workflow instances or third parties, it has to make sure that the user has given such consent. In a distributed environment, users interact with several parties. An interaction of the user with one party can trigger data processing in multiple components belonging to other parties. Nevertheless, users should only have to give consent once. The workflow management system is thus only responsible for requesting consent when one of its components triggers the processing and disclosure of identity information. When another party triggers the processing of identity information in the workflow management system, it assumes that the user has given consent. The underlying assumption is that a component that discloses identity information is responsible for ensuring the user has given consent. This does not include the further transfer of this data to other users or service providers.

D3 *Abstraction from technologies:* We need to distinguish between a concrete implementation and the generic FIM concepts. Consequently, we have to provide a concrete and a generic layer with a lean interface between them. Workflow-specific functionality should only work with the generic view of FIM. This makes it possible to use existing FIM libraries and update them when necessary to follow the evolution of the underlying specifications.

D4 *Standards-based architecture:* The architecture of our system and its interactions with the environment should be based on established standards. This includes the WfMC workflow reference model [Hol95] and the reference architecture of XACML [Mos05].

D5 *Clear assignment of functionality to components:* We want to assign responsibility for a security-relevant function to one component, as is the case for the specifications mentioned in D4. This reduces complexity and makes it easier to ensure the system behaves correctly.

D6 *Declarative configuration where feasible:* The workflow management system offers security functionality for applications with different security re-

quirements. Consequently, it must be possible to configure the system so that it fulfils the security requirements of a specific application. We see two alternatives to accomplish this: (1) The workflow management system provides operations that an application workflow can invoke in order to set configuration options based on its internal state. This requires explicit, imperative-style code in the application. In addition, the application bears the burden of keeping the configuration in sync with its internal state. (2) Declarative-style configuration accompanies the definition of application workflows. When performing security functionality, the workflow management system evaluates this configuration, taking the state of workflow instances into account. We prefer (2), as this approach allows for better separation of concerns, i. e. between the application functionality and FIM. In particular, this eases reuse of existing application workflows, since developers do not need to modify the workflow definition itself when configuring the security. This is in line with policy-driven security pursued by the WS-SecurityPolicy specification [NGG+07]. However, we expect that in some cases, it will not be possible to provide generic policy-based solutions, as application logic is closely combined with FIM functionality. For example, an application might select available offers based on attributes such as age. In such cases, an imperative approach is inevitable. In summary, we want to provide declarative configuration wherever possible, and interfaces abstracting from technical details otherwise.

We have formulated two kinds of design requirements. The architecture-related requirements are under our control insofar as they concern our workflow management system and its components. As such, the workflow management system envisioned must fully implement them. Fulfilment of the privacy-related requirements not only depends on our workflow management system but also on the environment it is running in and on the workflow-based application it executes. We therefore require the resulting system to do its part to fulfil them. We have already stated this assumption in general terms in D2. We will address the privacy issues of the various FIM concepts in Subsection 5.2.2, and the design decisions in Section 6.4 will take these issues into account. We will assess the resulting system with respect to the design requirements in Subsection 6.4.10 and state in more detail the assumptions regarding the environment so that the system as a whole can fulfil them.

5.2.2 Requirements Concerning Support for Individual FIM Concepts

Taking the design requirements from Subsection 5.2.1 into account, we now examine how to integrate different FIM concepts into a workflow management system. The discussion addresses the following aspects:

- *Data flow:* When implementing the FIM concept, on which occasions does identity information enter or leave the system? Which components of the FIM infrastructure are involved? This affects the system's interfaces.

- *Context used:* Which types of workflow context, such as the execution history of a workflow instance, have to be considered?

- *Lifetime:* What is the timeframe over which context and identity information have to be stored? The alternatives we see are the user's login session, the lifetime of a workflow instance, or a single activity. The answer affects where such information should be stored.

- *Need for configuration:* Does the functionality apply to workflows without explicit security configuration, or is it necessary to include a specific security configuration in each workflow definition?

- *Type of configuration:* Is entirely declarative configuration possible (cf. D6), or do workflow definitions have to include special activities that control the functionality? In the former case, we will have to define how to evaluate the configuration. In the latter case, we need to figure out how to provide the necessary information to workflow instances.

- *Basis for configuration:* Which elements of the workflow definitions must the configuration refer to, and what is the runtime information corresponding to these references? For example, a separation-of-duty constraint refers to activities in the workflow instance and the users that have performed or will perform these activities. The answer lets us decide which components should implement the functionality, and how they get the necessary information.

- *Privacy issues:* How can the workflow management system handle privacy issues (D1 and D2) arising with respect to the functionality in question?

By analysing each FIM feature with respect to these aspects, we have been able to develop the architecture presented in Section 6.4 in a systematic way. The resulting architecture and the way it implements FIM functionality is also in line with our design requirements, as we will demonstrate in Section 8.

The following subsections discuss the FIM concepts presented in Subsection 2.4.5.1 in the context of workflows. We have covered different ways that identity information can reach a relying party, i. e. the workflow-management system in our case. These are addressed in Subsection 5.2.2.1 (single sign-on) and 5.2.2.2 (incoming web-service calls). Further, we have addressed different ways of using this information, covered one by one in Subsections 5.2.2.3 to 5.2.2.5. Finally, we have addressed the combination of FIM and access control. Subsections 5.2.2.6 and 5.2.2.7 address this in relation to individual workflow activities and the entire workflow respectively.

5.2.2.1 Single Sign-on

This subsection discusses how to perform single sign-on for workflows' user interfaces.

- *Regarding single sign-on, when does identity information enter/leave the WfMS?*

  In a WfMS, users interact with the worklist handler to perform human tasks through a web-based interface. Identity information enters the WfMS when the user logs into the tasklist handler using single sign-on.

- *Which kind of workflow context needs to be considered when facilitating single sign-on for workflows?*

  At the time of single sign-on, information about workflow instances is not yet relevant. The information acquired through single sign-on is merely stored for later use.

- *What is the lifetime of context and identity information necessary for single sign-on?*

  The information acquired through single sign-on is needed at least for the duration of the respective single sign-on session. During this session, the user may view their task list and perform one or more tasks. When the user has performed a task, the WfMS creates a relation between the corresponding workflow activity and the user's identity information. The required lifetime then depends on the purpose the information is used for, as we will assess below.

- *What kind of configuration is necessary for single sign-on?*

  How the WfMS uses the information acquired depends on the tasks performed and the corresponding workflow definitions. We do not see any need, however, to configure how to perform single sign-on. This is because in our experience, single sign-on is a generic functionality that behaves the same for different applications.

- *Which privacy issues, if any, arise in connection with single sign-on?*

  Single sign-on causes the identity provider to pass identity information to the WfMS. This has obvious privacy implications. Therefore, the user must be able to trust the WfMS and be aware of any disclosure of their identity information. As long as the user does not perform any tasks, the WfMS does not need to provide any identity information to workflow instances. Of course, the user has to trust the WfMS itself not to disclose the information and to protect it against attacks. The WfMS must make clear which workflow instances identity information is potentially provided to when the user performs a task. Further, the WfMS has to provide access to the privacy policy of the workflow in question. On top of this, the WfMS could perform automatic checks based on a consent policy submitted by the user.

5.2.2.2 Incoming Identity-Web-Service Calls

In this subsection, we examine how the WfMS deals with web-service calls that it obtains.

- *Regarding incoming identity-web-service calls, when does identity information enter/leave the WfMS?*

  In the infrastructure envisioned, web-service calls can carry identity information pointing to the individual in whose name the caller invokes a service. Workflows orchestrate service compositions and can both invoke and provide web-service interfaces. When such an interface is invoked, the call contains identity information in addition to the payload.

- *What kind of workflow context needs to be considered when processing incoming identity-web-service calls?*

  Each web-service call invokes a specific activity in a specific workflow instance. The WfMS has to determine this activity based on information in the payload of the call. To this end, WS-BPEL specifies *correlation sets*. A correlation set is a named group of properties that identify a conversation. The values of these properties are initialised by the first message of the conversation. Subsequent incoming messages are then routed to the same workflow instance. The context needed here is the content of the correlation sets of the running workflow instances.

- *What is the lifetime of context and identity information necessary to process incoming identity-web-service calls?*

  The identity information included in the incoming call only relates to a single activity. Again, the lifetime required depends on the purpose the information is used for. For example, if the activity is part of a separation-of-duty constraint, the identifier is needed until all other activities that are part of the same constraint have been executed. If the identity information is only used for authorisation on the activity level, it is not needed afterwards. In any case, identity information is no longer needed once execution of the workflow instance has finished.

- *What kind of configuration is necessary to process incoming identity-web-service calls?*

  Correlation sets, which determine the workflow activity and instance a call is routed to, are a standard feature of WS-BPEL and are not security-specific. We do not see any need to configure the actual extraction of identity information from the incoming call.

- *Which privacy issues, if any, arise in connection with incoming identity-web-service calls?*

  When a call is received, workflow instances receive identity information directly. It would not be convenient for users if they had to access the WfMS and give consent. This means the caller has to make sure the user has consented before making the call.

5.2.2.3 Using Attributes for a Personalised Service

Workflows coordinate web services and human activities. In particular, they compose web services in order to provide a more complex service for a specific user. Depending on the application, the user's attribute values can help them customise the service to their needs. For example, a workflow for booking a rental car can exclude premium-category vehicles when the user's driving license is less than two years old. We now discuss how to facilitate such customisations.

- *When attribute values are used to personalise a service, when does identity information enter/leave the WfMS?*

  The WfMS needs to provide identity information relating to workflow activities (Sections 5.2.2.1 and 5.2.2.2) to the corresponding workflow instances.

- *What kind of workflow context needs to be considered when using attributes for a personalised service?*

  Identity information always belongs to a single user performing tasks in a workflow instance. A workflow instance can acquire different attributes for the same user connected to different workflow activities. For example, this can be the case when a user has used different means of authentication. Accordingly, the activity that has acquired the identity information is important too, not just the user it belongs to. Because identity information, and in turn the user it belongs to, can be uniquely identified through the activity which has acquired it, no further context information is required.

- *What is the lifetime of context and identity information necessary when using attributes for a personalised service?*

  A workflow instance can potentially need identity information acquired by a specific activity at any point in its lifetime.

- *Is configuration necessary to personalise a service based on attributes?*

  The actual customisations highly depend on the individual application. For example, one application might perform different functionality depending on attributes such as age or insurance status. Another application might include attribute values in calls to external services.

- *What kind of configuration is necessary?*

  The WfMS does not know how to process the attributes. This completely depends on the workflow definition. It is best specified using imperative workflow logic, as there are no generic patterns that can act as a basis for declarative configuration. The sole task of the WfMS is to provide user attributes to workflow instances, which process this information according to the workflow definition. We will assess different ways of providing attributes in Section 6.4. Configuring declaratively *which* piece of information to retrieve appears feasible. However, the resulting data needs to be inserted into the data flow of the workflow, which is connected to the imperative application logic. To this end, we see a need for explicit activities that retrieve identity information as specified declaratively and make it available to the application logic of the workflow.

- *What is the configuration based on?*

  It must be possible for developers of workflow applications to state *whose* attributes are needed. One way to accomplish this is by referring to an activity in the workflow, and thus the identity which performed this activity.

- *Which privacy issues, if any, arise in connection with attributes used for a personalised service?*

  A user must be aware of which attributes are provided to a workflow instance before they trigger the actual transfer. Given this, it is implied that the instance can use the attributes.

5.2.2.4 Recognising Users in a Stateful Interaction

Another feature envisioned, different from the previous one, is an application that interacts with a user over several steps of a workflow. The point here is not that the application has to behave differently for different users, but that it has to remember previous interactions with a user.

- *When recognising users in a stateful interaction, when does identity information enter/leave the WfMS?*

  When a user accesses the WfMS a second time, it has to recognise them based on the identity information provided.

- *What kind of workflow context needs to be considered to recognise users in a stateful interaction?*

  The WfMS must know the identifiers of users who have performed activities in the past, so that it can compare them to the identifier of the user performing the current activity.

- *What is the lifetime of context and identity information necessary to recognise users?*

  The interaction of an application with a user usually spans the entire lifetime of a workflow instance, so the WfMS needs to remember identifiers of past activities until the instance has terminated.

- *What kind of configuration is necessary to recognise users?*

  Applications have tasks that the same user should perform, so one must be able to specify which tasks these are. This is a kind of workflow-level authorisation constraint; we consider it below in Section 5.2.2.7.

- *Which privacy issues, if any, arise when recognising users in a stateful interaction?*

  With this functionality, application workflows can learn that a user is the one who has performed a previous activity. However, as this is necessary to maintain a stateful interaction, users can reasonably expect this type of use when they perform an activity and provide identity information to a workflow.

5.2.2.5 Invoking Services on Behalf of Users

In a framework based on identity web services, workflows invoke these services on behalf of a user. This enables them, for instance, to access personal data the user's personal data store. To this end, it is also necessary to find services available to this user, and select one. The framework has to provide a *discovery service* that finds services of a given type available to a specific user. The address of the user's discovery service is part of the assertion provided by the identity provider.

- *When invoking services on behalf of users, when does identity information enter/leave the WfMS?*

  When invoking services on behalf of users, the WfMS has to use identity information for two purposes: First, it has to discover services available

to a specific user. Second, it has to add credentials asserting that it is making the call on behalf of this user when it actually performs the call.

- *What kind of workflow context needs to be considered when invoking services on behalf of users?*

  The user on whose behalf the WfMS makes the call depends on the activity. For their credentials to be available to the workflow instance, they must have performed an activity in it before. The WfMS uses this identity information not only for the actual call, but also to perform service discovery. To this end, it contacts the user's discovery service. In summary, the context used by the WfMS is the activity performing the call, and a previously performed activity in the same workflow instance together with its identity information.

- *What is the lifetime of context and identity information necessary when invoking services on behalf of users?*

  The WfMS has to remember the credentials of users who have performed activities in the workflow instance.

- *What kind of configuration is necessary to invoke services on behalf of users?*

  A characteristic of workflows is that more than one user can be involved, and that activities triggered by one user might not be executed immediately but only after some condition is fullfilled. For instance, approval by another user might be necessary. Accordingly, it is not immediately clear on *whose* behalf the WfMS needs to make a call. For each outgoing call to an identity web service in workflow definitions, it is necessary to configure which identity information to use.

- *What is the configuration based on?*

  Outgoing calls use identity information that has entered the system in a previous activity. To specify which identity information to use, workflow designers thus have to refer to an activity.

- *What requirements for the configuration exist?*

  Referring to a past activity appears to be possible in a straightforward way by specifying the name of the activity. We deem this sufficient for a start, although in the case of conditional executions or loops, more sophisticated mechanisms become necessary. This is because in such a case, different activity instances with the same name are possible. We will leave solutions to this issue to future work.

- *Which privacy issues, if any, arise when invoking services on behalf of users?*

  When invoking services on behalf of users, this issue is twofold: First, the workflow transfers identity information to the services invoked. The WfMS alleviates the associated privacy risks by taking user preferences into account when choosing services. To achieve this, it determines the trust level of available services according to the user's policy. Second, the services invoked may expose the user's personal data. In principle,

the user consents to this by using the workflow, but appropriate control mechanisms are needed. However, note that this applies to any chaining of service calls and is not workflow-specific. In addition, the invoked service may check whether it trusts the WfMS to convey user consent correctly.

5.2.2.6 Activity-level Authorisation

This subsection examines how to perform authorisation, taking only the current activity into account. Attribute-based access control uses user attributes to decide whether access to a given entity should be granted.

- *Regarding activity-level authorisation, when does identity information enter/leave the WfMS?*

  In workflow management, attribute-based access control concerns human tasks as well as service calls directed to workflow instances in the name of a user.

- *What kind of workflow context needs to be considered for activity-level authorisation?*

  The WfMS has to use the attributes associated with the incoming call or provided via single sign-on to the worklist handler.

- *What is the lifetime of context and identity information necessary for activity-level authorisation?*

  The access control decision can be taken for each of these workflow activities individually. This means that identity information is used immediately for access control and is no longer needed afterwards.

- *Is configuration necessary for activity-level authorisation?*

  The attributes required depend on the application, so each workflow definition needs an access control configuration.

- *What is the configuration based on?*

  Workflow definitions must specify the attributes needed for each activity.

- *What kind of configuration is necessary?*

  A declarative specification such as an XACML policy is suitable for this purpose.

- *Which privacy issues, if any, arise in connection with activity-level authorisation?*

  We do not see any additional issues in this case. This is primarily because the WfMS does not need to provide any identity information to workflow instances.

5.2.2.7 Workflow-level Authorisation

Workflows are stateful and can involve several users performing tasks. Authorisation needs to take the relationship between tasks into account. This subsection therefore discusses how to support such authorisation methods in our architecture. The fundamental concepts that cover most application needs are separation of duty and binding of duty (see Subsection 2.4.3.2). Separation of duty requires different users for conflicting activities, such as *Authorise payment* and *Issue cheque*, while binding of duty requires the same user to perform several activities. We always apply these constraints to complete instances of workflows, i.e. the most refined business context, using [CXO+07] terminology. More complex relationships, e.g. based on attribute value, are possible, and can be implemented following the same basic approach.

- *Regarding workflow-level authorisation, when does identity information enter/leave the WfMS?*

  As in the previous subsection, workflow-level authorisation concerns human tasks as well as service calls directed to workflow instances. Authorisation decisions use identity information that has entered the WfMS for different activities, either through single sign-on or incoming web-service calls.

- *What kind of workflow context needs to be considered to perform workflow-level authorisation?*

  To facilitate separation or binding of duty, the WfMS needs to know who has performed past activities in the same workflow instance.

- *What is the lifetime of context and identity information necessary for workflow-level authorisation?*

  The information may potentially be needed throughout the lifetime of the workflow instance.

- *Is configuration necessary for workflow-level authorisation?*

  The separation- or binding-of-duty constraints that exist between the activities of a workflow are application-specific. Accordingly, workflow developers must be able to specify the constraints for each workflow definition.

- *What is the configuration based on?*

  Constraints specify activities that must be performed by different users (separation of duty) or the same user (binding of duty).

- *What kind of configuration is necessary?*

  A simple specification would just list activity names. However, this is not sufficient when activities are executed several times, e. g. in the case of loops. We will leave solutions to this issue to future work.

- *Which privacy issues, if any, arise in connection with workflow-level authorisation?*

The WfMS can only enforce separation- and binding-of-duty constraints correctly with persistent pseudonyms. These allow the WfMS to recognise users permanently.

5.2.3 Conclusions

We have systematically explored the implications arising from the integration of FIM functionality into a workflow management system, refining and extending the general security requirements from Section 5.1. In particular, we have discussed how the system processes identity information, and what context and configuration options are needed. This paves the way for us to extend a secure workflow management system by adding FIM support in Section 6.4.

5.3 Overall Summary

In this chapter, we have presented two different kinds of requirements: a broad range of general security requirements, and requirements specific to the integration of FIM functionality. Both encompass a range of functional and design requirements that the system in question must meet. We also explained our design requirements in line with this distinction. In the next chapter, we will design a system of this kind using the same two-step approach. First, we develop the general structure of a secure workflow management system, taking into account the requirements (both functional and design requirements) from the first part of this chapter. We then analyse how it can be extended by adding FIM support.

Chapter 6

System Design and Architecture

In this chapter, we design the architecture of a workflow management system that fulfils the requirements from the previous chapter[1]. First, we explain the types of configuration and workflow context the system has to handle. Following the same structure as for the requirements, we then address extensions for general security support, limiting ourselves to a subset of the requirements from Section 5.1. Our aim is a flexible architecture. The resulting architecture will thus be able to accomodate the remaining requirements, as we will assess in Section 8.2. After that, we address the integration of FIM functionality into the resulting workflow management system.

6.1 Security configuration

A general-purpose secure WfMS needs configuration to be able to implement the security requirements of different applications. In the following, we briefly explain from which sources such security configuration can be derived, and which types of security configuration there are.

6.1.1 Sources of security configuration

We see different ways to create the security configuration of a WfMS, i. e. a definition of how the WfMS has to behave to fulfill the security requirements of an application: (1) In some cases, default values resulting from general requirements are sufficient, e. g. all communication should be encrypted. (2) Other sources include the annotations defined by workflow-security modelling approaches like [WS07], [MTM09] and [RFMP07] (see also Section 2.3), and settings for the entire model, defined using a suitable language. (3) A security configuration can also be created at runtime by deriving it from user interactions, e. g. a user giving consent for the workflow to access their personal data, or by retrieving policies from an appropriate store, such as the trust policy of a user involved.

There may be several parties with different security requirements, resulting in several security policies. Think of a loan approval workflow in a bank where the risk assessment is outsourced. The bank is primarily interested in accurate risk predictions, while the customer prefers services with no privacy violations.

[1] The contents of this chapter have been published as part of [MMvSB10] and [MB14].

In addition, the legislator may only allow data transfer to service providers abroad when the respective country has sufficient data protection laws as well. When more than one policy exists, conflicts can arise. Such conflicts must be resolved; [CF09] describes an approach for this.

6.1.1.1 Types of security configuration

Another question is how configuration can be expressed. This issue is important in deciding which components are most suitable for this purpose. Again, we see alternatives:

- *Declarative policies:*

  For classic access control, languages like XACML are a solution. For other uses, like specifying trust, specialised languages exist [BDOS05, KNS05].

- *Simple variable:*

  In this case, certain settings are assigned values of basic data types, either for the entire WfMS or on the level of workflow instances. Examples includes turning encryption on and off, or setting the log level.

6.2 Security-relevant context

Our WfMS has to enforce security in a way that adapts to the situation by using context. For this reason, we now take a closer look at the notion of context.

Dey [Dey01] provides a definition of context useful for context-aware applications. He claims that existing definitions of context are too specific. Context is "all about the whole situation relevant to an application and its set of users," and it is not possible to enumerate the important aspects of the situation. He then defines context as "any information that can be used to characterise the situation of an entity". An entity in turn is "a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." A context-aware system is one that "uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task." Applying this definition to context-aware security enforcement for workflows means that context is information characterising the situation of entities considered relevant to the security properties of a workflow. Users' tasks are explicit in workflow definitions, so security enforcement can consider which tasks users currently perform.

Rosemann et al. [RRF08] investigate context with respect to business processes (workflows) in terms of flexibility. Accordingly, they see context as a stimulus for change. To come up with a definition of context that is operational, they develop a taxonomy for context that consists of four layers. The immediate context "covers those elements that directly facilitate the execution" of a workflow: data, resources, applications and events. The other layers (internal, external and environmental) represent increasingly abstract concepts.

For us, it is important that workflows provide an orchestration of entitities involved in a business interaction in order to achieve a certain outcome. While implementing the application scenarios, we encountered different kinds of context that can influence security enforcement:

- *Activity context:*

  This is context relating to particular calls (web-service calls or creation of human tasks), namely which activity in which workflow instance has caused the call.

- *Associated entities:*

  This context is entities like users or external data sources associated with a workflow instance. Examples are a *curriculum vitae* stored in an external e-portfolio and used in the workflow, or a user whose job qualities are assessed by the workflow. An entity can be associated with the entire workflow instance or some element of it. For instance, the user who has performed a given task is associated with the corresponding activity.

- *Execution state:*

  This type consists of information such as activities waiting for execution or the iteration of a loop that is currently executing.

The WfMS must have the necessary context information available to enforce security, an issue we will address below.

6.3 General Security Enhancements to the WfMS Architecture

In this section, we design an architecture extension based on the design-related and functional requirements from Section 5.1. The subset of functional requirements addressed here covers all security facets and is thus representative for the entire set of security requirements.

Regarding authentication, we only address the authentication of users (R8 to R11). The treatment of access control encompasses workflow activities (R12 to R15) and the corresponding constraints (R17, R18, and R21), as well as the selection of web services (R28 to R30). It skips reassignment of duty (R19 and R20), assignment strategies for activities (R12 to R15 as well, and R16 in particular), and the handling of data (R22 to R27). Further, non-repudiation and confidentiality of web-service calls (R31 to R34) are addressed, as well as confidentiality of data shown to workflow participants (R35). We postpone the problem of linking identities across workflow instances (R36) to the next section, which addresses the integration of FIM functionality.

The very general requirement regarding integrity (R37) is not addressed at this stage, as it can only be assessed sensibly when implementing the architecture. Security auditing (R38 to R41) will be the subject of Section 8.3. We address key management (R43 and R44) in the context of non-repudiation and confidentiality of web-service calls.

6.3.1 Approach for the Integration of Security Functionality

In Subsection 2.4.4 and Section 3.6 we presented approaches for integrating non-functional aspects in general and security in particular into workflows. The next question is how to best integrate the security functionality demanded by our security enhancements to the architecture of a workflow management system.

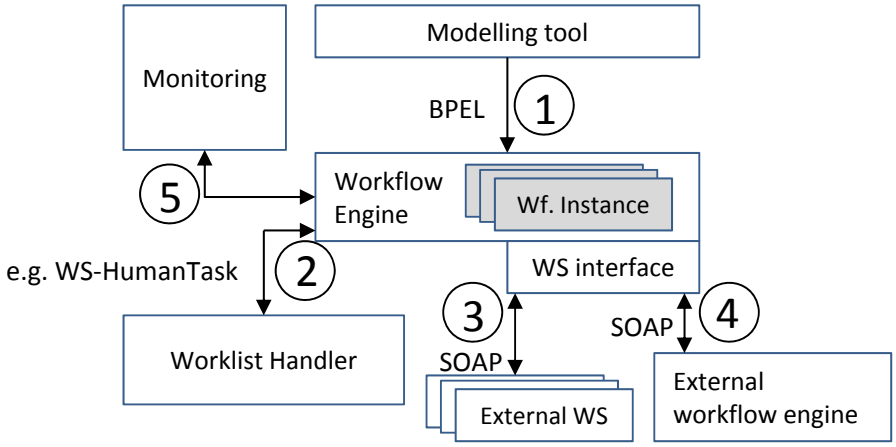


Figure 6.1: Classical architecture of a WfMS in SOA.

We see three fundamentally different ways of weaving security aspects into the workflow.

- *Security functionality is implemented as part of the workflow engine.*
- *Security functionality is implemented as a separate component or several separate components.*
- *The workflow is transformed so that the transformed workflow performs security functionality itself.*

Alternative 1 should be used only when absolutely necessary, as it conflicts with the design goal *Minimal Changes*. Alternative 2 is appropriate when the aspect itself is best expressed as a workflow or a fragment of one. It is unsuitable when the natural form of the aspect is different, e. g. declarative. In such cases, Alternatives 1 and 2 can be used. They are even necessary when the security functionality cannot be specified as a workflow in an unmodified workflow engine. In contrast to Alternative 1, Alternative 2 allows the underlying concept to be clearly represented more clearly by encapsulating it in a dedicated component and separating it from the normal workflow execution. In order to weave security aspects implemented by separate components into the execution of workflow instances, it is necessary to transform or instrument workflow definitions (i. e. use Alternative 3).

6.3.2 Security requirements and their potential implementation

In the following, we go through different kinds of functionality necessary for a secure WfMS in service-oriented architectures. As indicated above, we have selected a subset of the requirements from Subsection 5.1.2 that allows us to explain the general anatomy of our architecture. This subset covers most of the security frameworks listed in ITU-T Recommendation X.810 (cf. Subsection 2.4.1.3). When addressing distinct features that a secure WfMS must

possess, we discuss alternatives for their implementation and how they affect the architecture and the interfaces of a WfMS according to the workflow reference model (Fig. 6.1).

6.3.2.1 Encryption and signing of web-service communication

Description of requirement The WfMS has to encrypt web-service messages so that only the recipient can decrypt them, and to sign these messages so that the recipient can determine who has issued the request, i. e. which provider or workflow has done so in the name of which user. The bare message exchange can be accomplished by WS-Security (cf. Subsection 2.4.4).

Necessary information To fulfil this requirement, the WfMS must have access to the communication partners' public keys, and private keys to use itself.

Parts of the architecture affected As this requirement concerns communication with external services, both the workflow engine and its web-service interface (interfaces 3 and 4) are affected: They must support the cryptographic protocol extension and allow the setting of parameters, like the keys to be used.

Implementation We see several alternative implementation routes. One is by adding support for the protocol extension to the workflow engine. Another is introducing a new component that acts as a proxy between the workflow engine and the external web services called. A modification of the workflow engine would be specific for the combination of a given engine and a given encryption protocol. When using a proxy, a transformation component must instrument the specification of the workflow to use the proxy, and the workflow engine must ignore messages from other sources than the proxy. We advocate the use of a proxy, as this works with different workflow engines and it can be replaced to support different encryption protocols. By way of example, a call `ws.function(payload)` becomes `pep.callWS(wstype, function, payload, instance, activity)`.

6.3.2.2 Discovery of trustworthy web services

Description of requirement Service discovery is the search for and selection of services. In our context, the goal is to find services that are trustworthy, based on specifications by the application and the users involved in the workflow. The context determines whose trust policy is relevant. This tends to be the user whose data will be transferred to the service in question. There might be more than one trustworthy service according to the policies applicable. In this case, if the user so desires, they should be able to explicitly choose one of these services. As long as the workflow has not called the service, the user should be able to change the selection.

Necessary information To discover trustworthy web services, the trust policies of the workflow itself and the user are required. In addition, the WfMS must receive the user's decision and remember a selection when the interaction spans several calls, e. g. when several data items are stored in a repository.

Parts of the architecture affected This requirement affects the workflow engine and its web-service interface (interfaces 3 and 4), and the engine's interaction with the worklist handler (interface 2) to fetch a decision from the user.

Implementation There are two parts to the implementation of this feature: (1) The WfMS must manage the workflow's web-service communication as follows: It must discover available services and select one that fulfils the trust policies applicable, store the selection, and route calls to the service selected. We see two options for doing this: (a) It is possible to implement this almost entirely by transforming the workflow. The only exception is the management of keys and other encryption parameters for the selected services. This is because an unmodified workflow engine does not necessarily support the respective protocol. (b) Implementing this step in the proxy discussed above. – We deem (b) superior, as it brings together the necessary functionality in one place together with other functionality used to call web services. It also makes it easier to recheck trustworthiness directly before future calls. Workflow instrumentation allowing use of the proxy is alredy required to support encryption and signing. (2) The user must be involved in the process of service selection. As the necessary user interactions can be implemented as human tasks, the worklist handler manages them. The most natural way to express the flow of such a user involvement (an *advice* in the terminology of aspect-oriented programming), including user interactions and calls configuring the proxy, is a workflow fragment. Consequently, we argue that the preferred solution is to weave the advice into the workflow. Implementation in the workflow engine itself is not preferred, according to our design goal of *Minimal Changes*. However, changing the service selection (an asynchronous user interaction) requires a differently-designed user interface which has to show the user the state of instances and let the user make changes. The interface then applies the changes via the so-called UInvolve Handler without the participation of the workflow engine.

6.3.2.3 Access control for human tasks

Description of requirement It is necessary to control who performs the human tasks in a workflow. Decisions are either specific for one task and based on the properties of the user performing the task (such as their job function) or take the history of the workflow instance into account as well. The latter holds for stateful authorisation constraints (Subsection 2.4.3.2).

Necessary information Implementing this requires information about users, such as their role or attributes, about the history of the instance (who has performed which human tasks in this instance?), and the context of human tasks, namely which activity in which instance has caused them.

Parts of the architecture affected This feature affects the interface of the workflow engine to the worklist handler (interface 2), as context must be available. It also affects the worklist handler itself, as the handler must evaluate whether a user logging into it may perform the human tasks available.

Implementation The functionality required consists of two parts: (1) The worklist handler must check permissions when a user retrieves the list of tasks available to them and again when they try to perform a task from this list. This second check is because the user might no longer be eligible, e. g. when another user has performed a task in the meantime, connected by a BoD constraint. The natural form of access control rules and constraints is a declarative policy. So including access control directly in the workflow via instrumentation is not a desirable option. In addition, this would make it hard for the worklist handler to recheck permissions when the task is actually performed. The worklist handler needs an interface to a component that can make access control decisions. However, this functionality should not be implemented in the worklist handler itself. This is because keeping the necessary history would require substantial extensions. Thus, a separate component that has access to the history would appear to be a better solution. (2) The history of human tasks performed must be captured. We see three alternatives: (a) using a proxy between the workflow engine and the worklist handler, (b) instrumenting the workflow, or (c) modifying either the workflow engine or the worklist handler. We advocate (a) because a separate component is needed anyway to make access control decisions, and because the worklist handler can retrieve the history information whenever required. Instrumentation is still needed, though, to submit the necessary context to the proxy component when creating human tasks. Instead of `wlh.createTask(taskDef, payload, allowedRole)`, the workflow now has to call `pep.createTask(taskDef, payload, instance, activity)`. The parameters `instance` and `activity` make up the context of the call. `payload` is the body of the message; its meaning is application-specific. Because the PDP-Wf can determine eligible users using the context by evaluating a policy, the parameter `allowedRole` is no longer necessary.

6.3.3 Summary of necessary extensions

In summary, the following functionality must be added to the WfMC workflow reference model:

Web-service communication Here, the WfMS needs to support protocol extensions for encrypted web-service calls with cryptographic signatures, perform service discovery and keep track of the results, route web-service calls to the service selected, and enforce that these services adhere to the trust policies applicable.

User interactions and human tasks The WfMS needs to interact with users to confirm the selection of web services or to convey consent to the processing of personal data. It must support both synchronous and asynchronous user interactions. For human tasks, it must be able to make authorisation decisions while taking the history into account.

Handling context information The WfMS must capture and store context information about workflows, and different components need to access this information. As a minimum, it must be known which activity in which instance of a workflow has invoked an operation. The WfMS must also keep state

information, including history information about who performed human tasks, the users assigned to human tasks that are still due, and selected web services.

6.3.4 Resulting Architecture

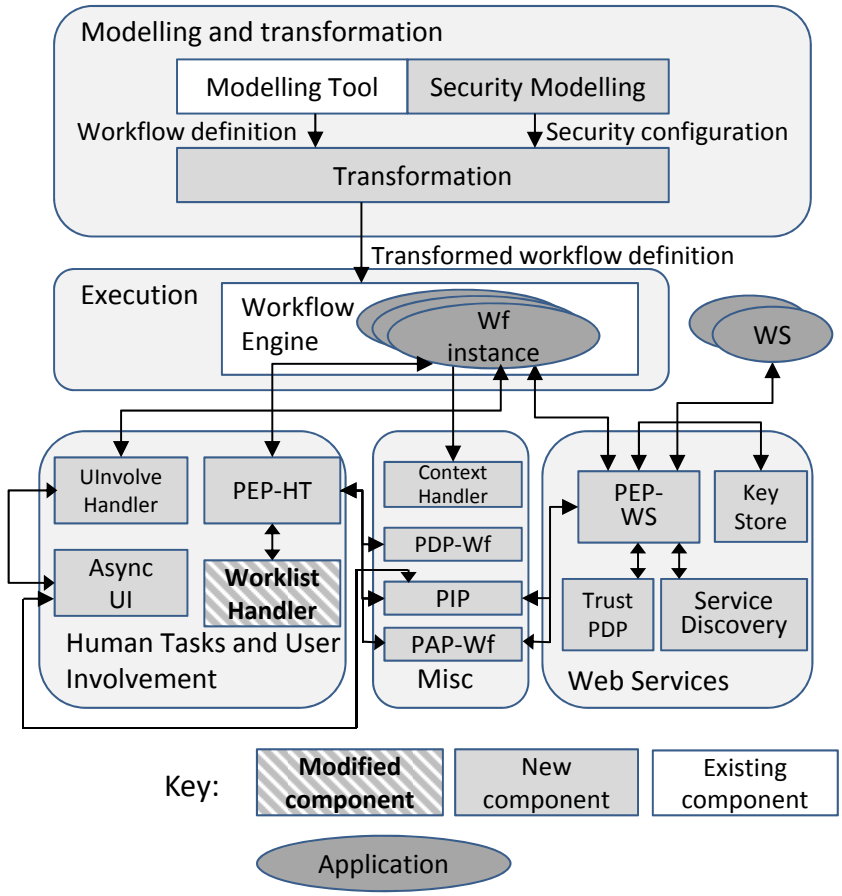


Figure 6.2: Resulting architecture

Figure 6.2 shows the architecture we are proposing. We will now explain how it provides the functionality just described.

Handling web-service calls A new component, the Policy Enforcement Point for Web Services (PEP-WS), performs all security-related tasks in connection with web-service calls. It acts as a proxy for these calls, i. e. the workflow sends these requests to the PEP-WS instead of the external web service. The PEP-WS then checks whether the service is still trustworthy according to the applicable policies, encrypts and signs the request with the correct keys, and finally sends it to the selected service (Algorithm 1). Thus, the PEP not only enforces policies, but also acts as a protocol adapter. Handling web-service calls involves the PEP-WS, the Key Store, the Trust PDP (all shown

Algorithm 1 PEP-WS handling a web-service call

```
1: ws ← PIP.getSelectedWS(activity, pid)
2: policies ← PIP.getApplicablePolicy(activity, pid)
3: trusted ← TrustPDP.isTrustworthy(ws, policies)
4: if trusted then
5:    key ← KeyStore.getKey(ws)
6:    msg ← encryptAndSign(request, key)
7:    reply ← ws.sendMessage(msg)
8:    if isSignatureValid(reply, key) then
9:       return decrypt(reply)
10:   else return error
11: else return error
```

in the box *Web Services* in Fig. 6.2), the PIP, and a workflow modified by the Transformation component.

Handling human tasks To handle human tasks, we propose another new component, the Policy Enforcement Point for Human Tasks (PEP-HT). Like the PEP-WS, it acts as a proxy between the workflow and the worklist handler. Before passing on a request to create a human task, it stores context information in the PIP. When a task is completed, the PEP-HT stores this information in the PIP as well, before returning the result to the workflow. We propose modifying the worklist handler so that it retrieves an authorisation decision from the PEP-HT before including a human task in a user's worklist or letting the user perform the task. The PEP-HT calls the so-called Workflow Policy Decision Point (PDP-Wf) to make the decision, using history information from the PIP. The PDP-Wf is a special PDP able to evaluate history information. It evaluates the appropriate policies, thereby making the reason for the decision explicit. Handling human tasks involves the PEP-HT, a modified Worklist handler (in the box *Human Tasks and User Involvement* in Fig. 6.2), the PIP, the PDP-Wf and a workflow modified by the Transformation component.

Keeping persistent information There are two kinds of security information that must be retained across interactions. (1) The Policy Information Point PIP stores associated entities, namely the web services selected, the workflow models the instances belong to (to determine the policies applicable to the instance), human tasks waiting to be performed and the context of their creation, the history of who has performed which tasks, and user attributes. It also stores variables (Section 6.2) defined for workflow instances. (2) When services are discovered, the Service Discovery component also has to supply the keys used to communicate with them. The PEP-WS stores these in a new component, the Key Store. – The Worklist handler is stateful as well, remembering the human tasks waiting to be performed, but this information is not security-specific. Thus three components – the Worklist handler, the PIP and the Key Store – have to keep persistent information of different kinds.

Synchronous user interactions Many user interactions, such as asking a user to select a service, are similar to application workflows and thus best

expressed as workflow fragments. In addition, they need to be performed at defined points within the workflow. Accordingly, we use the Transformation component to include appropriate fragments in workflow definitions before deployment. In AOP terminology, these are advices implemented in the component language. The user interactions orchestrated by such advices can change the configuration of security components. For example, when the user selects a service, this information has to be stored in the PIP. We provide the UInvolve Handler as a single contact point for the workflow. It receives requests to change the security configuration from the workflow. An example is the registration of a service selected by the user. The UInvolve Handler checks whether the configuration change adheres to the policies applicable and passes it to the right component, in this case the PIP. Thus, synchronous user interactions require a workflow instrumented by the Transformation component, nearly all components from the box *Human Tasks and User Involvement* (UInvolve Handler, PEP-HT and Worklist handler), and the PIP.

Asynchronous user interactions For asynchronous user interaction, such as revoking consent, our architecture includes a separate component (Async UI). This modifies the configuration of security components via the UInvolve Handler.

Providing context information about workflow instances We propose several ways to handle context information, depending on its type: (1) Activity context is transmitted directly from the workflow to the PEP-WS or PEP-HT with the corresponding call; see the interface extension from Sections 6.3.2.1 and 6.3.2.3. (2) Entities associated with an instance include users who have performed human tasks. The PEP-HT stores this information, which includes services selected by the user, in the PIP. The workflow invokes the UInvolve Handler to register the selection, and the UInvolve Handler stores the selected service in the PIP as well. (3) To keep our architecture extensible, we propose a component that can make available other kinds of context: the Context Handler. In all three cases, the Transformation component instruments the workflow so that it provides the context to the other components. The PEP-WS and PEP-HT handle context necessary to fulfil their tasks, while the Context Handler provides the execution state, which can be used for access control decisions.

Policy evaluation Our architecture has to deal with different types of policies. For human tasks, the authorisation decision depends on the history of the workflow instance, i. e. who has performed other tasks. It is based on polices that can express history-based constraints. Service selection relies on trust policies that yield a trustworthiness score. The PDP-Wf and the Trust PDP handle these kinds of policies respectively. In addition, it is conceivable that other policies apply to the actions performed by a workflow, e. g. legal regulations allowing the processing of sensitive information for certain purposes only. A Master PDP can combine the results from the evaluation of different policies and handle possible conflicts [CF09]. It acts as a wrapper for different PDPs, including the PDP-Wf and Trust PDP. By using policies tailored to particular domains, the structure of security specifications can be preserved. Such policies

are stored in the PAP-Wf (Policy Administration Point for Workflows) when they are specific to workflow models. To summarise, we propose two special kinds of PDP and and the possibility to introduce additional PDPs by using a Master PDP to aggregate policy decisions.

Summing up, we require changes to legacy components only where necessary. The architecture has evolved from an established reference model and allows integration with existing security infrastructure in areas such as authentication or auditing. Like the XACML reference architecture, it contains separate components for policy decisions, policy enforcement, and providing context needed for the evaluation of policies. The structure of security specifications has an explicit representation in the components proposed. We argue that our architecture meets the design goals from Section 5.1.5.

6.4 Design of a System Integrating FIM Functionality

Building upon the security extensions for a WfMS which were developed in the previous section, we pursue the integration of the WfMS with an infrastructure for federated identity management in this section. In detail, we investigate how a secure WfMS can support FIM functionality, fulfilling the requirements from Subsection 5.2.2. This section is structured as follows: First, we examine which functionality a generic FIM layer can perform, in line with D3, and how WfMS components can use this layer. In line with D5, we then introduce new components of an extended WfMS responsible for acquiring identity information, storing it, and performing functionality based on it, such as access control. This includes revisiting design decisions already adressed in Section 6.3 with respect to the architecture not including FIM functionality. We define the interactions between the components, in particular, how workflow context reaches the components that need it. We also describe the configurations necessary for workflows running in the extended WfMS.

6.4.1 Basic Architecture

The basis for the extensions presented in the following is the WfMC reference model presented in Subsection 2.3.1.

This model requires several kinds of extensions: Interfaces 2–4 concern the communication of workflows with the outside world at runtime. Workflows send and receive web-service calls through Interfaces 3 and 4. They create instances of human tasks in the worklist handler through Interface 2 and get notifications when instances have been performed. One has to extend the worklist handler and the web-service interface so that identity information is acquired. The core WfMS has to process that information to provide the FIM functionality described in Section 5.2.2. Interface 1 concerns the deployment of workflow definitions. Security configurations must accompany them, so it is necessary to define the structure of this configuration and how it is distributed to the components that need it.

We will extend this architecture in line with the design decisions explained in the following subsections. Figure 6.3 shows the extended architecture. New WfMS components have grey backgrounds. The diagram is structured into several parts: The main part of the WfMS, the engine that executes workflow

instances, is at the top. Below are the other components, including our new security components. The left part contains components specific to the handling of human tasks. The right part in turn deals with web services. The space in between contains components common to both groups. Components that are not part of the WfMS but of the surrounding infrastructure have a dashed edge and are shown at the bottom of the diagram.

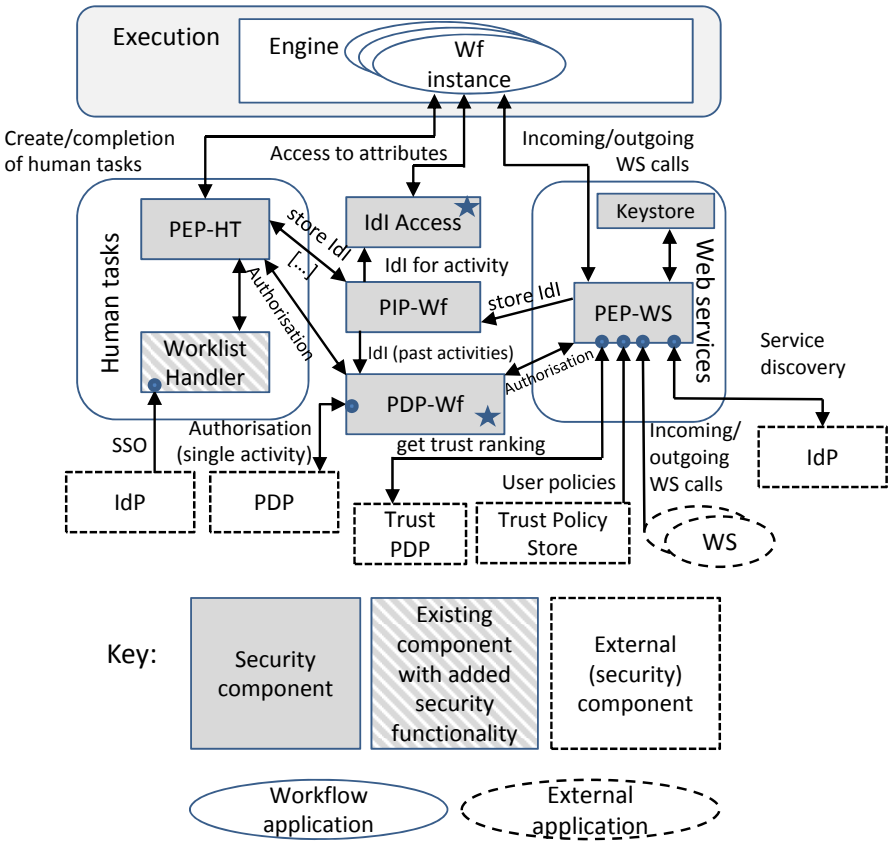


Figure 6.3: Architecture of a WfMS with FIM Support

6.4.2 Encapsulation of Technology-specific FIM Functionality

Problem In order conform with D3, technology-dependent FIM functionality should be encapsulated in a separate layer. The WfMS envisioned relies on different kinds of FIM functionality. We can partition the functionality required depending on how it deals with identity information, namely acquiring, outputting or accessing it: (1) Acquisition of identity information: Technically, identity information is represented as SAML assertions. The WfMS acquires it either through SSO in the case of web-based interactions, i. e. a user performing human tasks through the worklist handler, or when it receives identity-web-service calls through its web-service interface (Sections 5.2.2.1 and 5.2.2.2).

(2) Outputting identity information: The WfMS has to use identity information when invoking web services on behalf of users according to ID-WSF, and when requesting activity-level authorisation decisions using [AL05] (Sections 5.2.2.5 and 5.2.2.6). The exact behaviour for all this depends on the technical specifications mentioned. (3) Accessing parts of identity information: For some tasks, the WfMS itself needs to access parts of identity information. In particular, it needs to access attributes to provide a personalised service, and identifiers to recognise users in a stateful interaction and perform workflow-level authorisation (Sections 5.2.2.3, 5.2.2.4, and 5.2.2.7). Consequently, we need an FIM-layer implementation that can acquire and output identity information while encapsulating the implementation details, and that allows access to attributes and identifiers. Because several components of the WfMS need access to this functionality (as we will explain below in more detail), they must be able to share identity information stored in the FIM layer.

Alternatives It is possible to either implement protocol-specific *ad hoc* where it is required, encapsulate it in a library of our own, or use an existing library.

Discussion Using a library that handles the protocol-specific implementation details leads to better encapsulation. On the one hand, using an existing library might require some trade-offs to make the other parts of the implementation compatible with it. On the other hand, it will provide a stable ground for dealing with the protocols in question. Implementing our own library would require considerable effort, weakening the focus on the technology-independent features of the architecture.

Conclusion We choose to use a library encapsulating the technology-specific details of the implementation. While an existing library may have some shortcomings that influence implementation decisions, the alternative – implementing a library supporting distributed setups from scratch – is impractical because of the amount of work it would require. Therefore, an existing library should be used. The choice of a particular library will be made in Section 7.2.

Impact on the architecture For external communication based on SAML and ID-WSF, several components have to use the library implementing these protocols. This is indicated in Figure 6.3 by a small circle at the respective end of an arrow. In addition, these components need access to identity information (identifiers or other attributes) handled by the library. Components are marked with a small star to indicate this.

6.4.3 External Communication and Acquisition of Identity Information

Problem FIM functionality is visible at the external interfaces of the WfMS, namely the user interface for humans and the web-service interfaces. The WfMS has to use special protocols here to acquire and eventually output identity information. It also has to prevent unauthorised access to workflows. The question now is where to perform this functionality.

Alternatives The first option is to use proxy components. All external communication of workflows would pass through these proxies. They can selectively block messages, add or remove data such as identity information, and translate messages to and from protocol-specific formats. Alternatively, workflow definitions can be instrumented so that workflows themselves perform this functionality. We have presented these different approaches for integrating non-functional concerns into workflows in Subsection 6.3.1.

Discussion Requiring workflow instances to handle the respective protocols would require complex, technology-specific instrumentations. By contrast, assigning this functionality to dedicated components makes it easier to change the underlying protocol and to delegate technology-specific funcionality to ZXID, the library we have decided to use in Subsection 6.4.2.

Conclusion We have decided to have two dedicated proxies to isolate the workflow engine from the worklist handler and from external web services respectively.

Impact on the Architecture The two proxies are called *Policy Enforcement Point for Human Tasks* (PEP-HT) and *Policy Enforcement Point for Web Services* (PEP-WS). This is because they can enforce security decisions, as we will explain below. They are shown in the respective blocks of the architecture diagram.

6.4.4 Access Control

In Subsection 6.4.3, we already introduced components that intercept all communication between the WfMS and its environment. These proxies can selectively block messages for security reasons. In this subsection, we now have to take several design decisions concerning the access control part of our architecture. This concerns the partitioning of the system as a whole into components for policy decisions and policy enforcement, for human tasks and web services, and for activity- and workflow-level authorisation constraints.

6.4.4.1 Decisions and Enforcement

Problem We have to decide whether to separate access control decisions and enforcement.

Alternatives We can integrate access control decisions into the proxies already introduced or assign this functionality to one or more separate components.

Discussion The XACML reference architecture distinguishes between access control *decisions* and their *enforcement*, and assigns these decisions to distinct components, a *Policy Enforcement Point* (PEP) and a *Policy Decision Point* (PDP). The latter takes access control decisions by evaluating declarative policies. This distinction is in line with D4 and D5.

Conclusion We have decided to assign access control *enforcement* and policy-based access control *decisions* to separate components.

6.4.4.2 Decisions for Different Kinds of Activities

Problem The question is whether to use the same PDP (or the same set of PDPs, cf. Subsection 6.4.4.4) for different kinds of activities (i. e. human tasks and incoming web service calls), or different ones for each of these two activity types.

Alternatives One alternative is to use the same PDP(s) for both kinds of activities. In this case, the (separate) PEPs either have to use the same request format; if they use different formats, the PDP needs to understand both formats. The other alternative is to use different PDPs.

Discussion For both kinds of activities, access control decisions are based on the attributes and identifiers of the users attempting to to perform them. Moreover, workflow-level authorisation constraints are not limited to a single kind of activity. Thus, the same information needs to be provided to the PDP by the two PEPs, and the PDP functionality for both kinds of activities is the same.

Conclusion We will use one combined PDP for all activities.

6.4.4.3 Structure of Policies

In Subsection 5.2.2.6 and 5.2.2.7, we addressed the need to authorise the execution of activities both on the activity and workflow level. The WfMS should perform both kinds of authorisation based on policies. Depending on the structure of the policies, we see several options with respect to the components responsible for evaluating the policies.

Problem How should policies be structured with respect to activity- and workflow-level constraints?

Alternatives One option is to separate the policy into two parts. The alternative is to allow the mixing of two kinds of constraints. Both options allow the same constraints to be expressed, but the latter makes it possible to group constraints of both kinds. Application developers may find this more intuitive. Workflow-level constraints are expressed as predicates over tuples of activities. We have to decide what kind of constraints to allow.

Discussion The respective parts of the security specification are usually independent; see Chapter 8 of [BMPS09]. Having two independent policy parts allows their independent evolution. This is particularly important as languages for stateless activity-level policies are widely established and not workflow-specific, whereas languages for workflow-level constraints are not yet standardised to the same extent. Most approaches for workflow-level constraints use only the identifiers of users performing activities, not their attributes. It is

possible to keep a history of previous access control decisions and refer to it in policies. The ITU-T access control framework [ITU95a] specifies the concept of *Retained Access Control Decision Information (Retained ADI)*. [CXO+07] provides an implementation, but it uses an extension of PERMIS instead of XACML. Even with such support in the policy language, this approach requires specifying each constraint in the policy multiple times for each activity concerned. As a result, these constraints become less clear and it is more difficult to change the policy.

Conclusion We will use policies with two separate parts. We deem workflow-level constraints that do not take attributes into account sufficient. In particular, we support the predicates $=$ and $\neq$, which represent binding of duty and separation of duty.

6.4.4.4 Structure of the Policy-decision Point

Problem Having decided to separate the activity- and workflow-level parts of policies, the question now is how to assign the evaluation of these parts to components.

Alternatives We have to decide whether to implement a monolithic PDP or use separate components responsible for the two different parts of the policy.

Discussion The evaluation of activity-level constraints is stateless. By contrast, workflow-level constraints take the execution history into account. On the one hand, a monolithic PDP can have better performance, as it only needs to access identity information once and saves communication overhead. On the other hand, a modular PDP is more flexible: First, the policy language can be changed. Second, the stateless part of the policies heavily depends on the specific technology used. For example, when identity information is provided as SAML assertions, a policy can refer to the authentication method used, which is expressed in a way specific to SAML. Being able to plug in an existing implementation is favorable with respect to D3.

Conclusion We have chosen a modular implementation, because we deem flexibility more important.

Impact on the Architecture There are two PDPs in our architecture: (1) A PDP for the evaluation of workflow-level constraints, called *Policy Decision Point for Workflows* (PDP-Wf). See the middle part of the architecture diagram. (2) For the stateless part, any existing PDP compliant with the SAML profile for XACML [AL05] can be used. – The PDP-Wf uses ZXID library functions to invoke a stateless PDP, shown at the bottom of the diagram, using the SAML/XACML profile. ZXID can be configured to automatically relay the attributes acquired through SSO or an incoming ID-WSF-compliant web-service call to the external PDP. Both PEPs request access-control decisions from the PDP-Wf.

6.4.5 Connecting Identity Information with Activities

Identity information, whether acquired via SSO or incoming web-service calls, is connected to a specific execution of an activity in a specific workflow instance. The WfMS uses it immediately to authorise this activity (Sections 5.2.2.6 and 5.2.2.7), but also needs it later on, e. g. to invoke services on behalf of the user or for workflow-level authorisation of other activities. In this subsection, we decide where to store the relationship between activities and identity information, and how to establish this relationship in the first place.

6.4.5.1 Storing the Relationship between Identity Information and Activities

Problem Which component or components should store the relationship between identity information and activities?

Alternatives This relationship can be stored separately or together for human tasks and web-service calls.

Discussion On the one hand, establishing the relationship between identity information and an activity works quite differently for human tasks and web-service calls, as described below. However, this does not preclude storing the information in one place once the relationship has been established. On the other hand, workflow-level authorisation constraints can involve activities of both kinds. It is easier to evaluate them when the necessary information is stored in one place.

Conclusion We have decided to use one component for storage, as the information for human tasks and web-service calls must be combined anyway to evaluate workflow-level constraints.

Impact on the Architecture The new component is named *Policy Information Point for Workflows* (PIP-Wf), shown in the middle part of the architecture diagram. It provides identity information belonging to past activities to the PDP-Wf. Other components submit these relationships to be stored in it, as explained below.

6.4.5.2 Human Tasks

In a traditional WfMS, human tasks are executed as follows: An activity in a workflow instance creates a task instance by sending a request to the worklist handler. When a user has completed the tasks, the worklist handler sends a response to the workflow instance. Our envisioned WfMS in turn has to handle the identity information of users acquired through SSO, and has to decide whether a user may perform a task based on that information.

Example 19 (Creation of a human task instance)*: In our travel booking example, the request to create a human task comprises the following: The name of the human task definition ("Travel Authorisation"), the payload for the human task (i. e. the name of the traveller, the destination and date*

of the trip, and the reason for the trip), an endpoint for the callback such as `http://bp-engine.example/ode/processes/TravelBooking/AuthoriseTravel.Callback/`*, the name of the activity ("Authorise Travel"), and the ID of the workflow instance (e.g. "BP2342").*

Problem Establishing the relationship between identity information and activities has two facets. The first is how to establish this relationship in order to perform access control for the current activity. This includes defining which components are involved in access control. The second concerns establishing this relationship before it is stored in the PIP-Wf. These facets are independent, as in the first case, it is not yet clear whether the activity will actually be performed. Moreover, it is possible that different components will establish the relationship in the two cases.

Alternatives One possibility is that the worklist handler does not directly access any component of the WfMS except for the PEP-HT. This means it sends all access requests to the PEP-HT, which in turn forwards them to the PDP-Wf. The PEP-HT also registers completed human tasks in the PIP-Wf. The alternative is that the worklist handler *does* perform some or all of these calls itself.

Discussion Both alternatives require changes to the worklist handler. However, decoupling it from the PDP-Wf and the PIP-Wf requires changes that are less specific to our architecture. In any case, we have to extend the worklist handler to support SSO and request an access control decision before allowing users to see or perform tasks. Were the worklist handler to call the PDP-Wf directly, it would have to know which activity in which workflow instance had created a task, and support the protocol for such requests.

Conclusion We have decided to extend the worklist handler only minimally and assign additional functionality to the PEP-HT. This means that the PEP-HT forwards access control requests and registers completed tasks.

Impact on the Architecture Using ZXID extends the worklist handler to include SSO functionality, as indicated by a circle in the diagram. For each human task instance, the PEP-HT creates an ID, stores it and the endpoint reference of the callback in the PIP-Wf, and forwards the request to the worklist handler. Users can log into the worklist handler once using SSO and perform multiple tasks. Here, authorisation has to occur twice: When a user views their worklist, the worklist handler must know whether to include a given task. When a user actually performs the task, authorisation is required again because of constraints. Think of a BoD constraint between two tasks. As soon as a user has performed one of the two tasks, other users are no longer allowed to perform the remaining task. Whenever a task is completed, the worklist handler includes the correspoding ZXID session ID in the response. The PEP-HT stores this in the PIP-Wf before forwarding it to the workflow instance. The architecture diagram shows an arrow from the PEP-HT to the worklist handler. It stands for the creation and completion of human tasks and for authorisation requests. The PEP-HT is also connected to the PDP-Wf (for

authorisation requests) and the PIP-Wf (to store which identity information belongs to completed tasks).

6.4.5.3 Web Services

For incoming web-service calls, the WfMS has to perform access control before the calls trigger an activity in a workflow instance.

Example 20 (Incoming web-service call requiring authorization)*: We now assume that managers do not approve trips by human tasks. Instead, there is a dedicated application where they can view which employees are absent due to business trips, annual leave, illness, etc. The travel approval workflow notifies this application of new requests for business trips using a web-service call. Managers view the list of open requests at some later point, and approve trips based on the overview provided by the application. The application sends the results back to the workflow as a web-service call, using the credentials of the manager who approved the request. If these approvals were not subject to access control, it would be easy to send fake approvals to the travel booking workflow, allowing business trips to be booked without approval.*

Problem As explained in Section 5.2.2.2, BPEL provides correlation sets to determine the context of the call, i. e. the activity and workflow instance the call is directed to. Normally, it is the workflow engine that performs correlation. However, the WfMS needs this context to perform authorisation, and to establish the relationship between identity information and an activity, in order to store this relationship in the PIP-Wf.

Alternatives We see several approaches for addressing this problem: (1) Reimplementing correlation in the PEP-WS. (2) Instrumenting the workflow, especially activities receiving calls: The idea is to insert a workflow fragment that sends the ID of the workflow instance and the name of the receiving activity to the PEP-WS. With this information, the PEP-WS can perform authorisation and send the authorisation result as a reply. If authorisation is denied, the received message is discarded, and the receiving activity is started again. (3) Close integration of the PEP-WS with the workflow engine to perform correlation without actually delivering the message. (4) A minimal solution supporting only calls that start new workflow instances. For such calls, no correlation is necessary. However, the PEP-WS must learn the ID of the newly created instance. This is possible using a simple workflow instrumentation.

Discussion We deem (1) impractical. It would require reimplementing a substantial part of the functionality of the workflow engine in the PEP-WS. To accomplish this, the PEP-WS would need workflow state currently not available to it, such as the activities waiting for calls. (3) makes our extensions contingent on a particular workflow engine, contrary to our design objectives. (2) is reasonable. (4) is suitable for a large class of applications and is easiest to implement.

Conclusion Our pragmatic solution is to start with (4) and leave (2) as future work. Subsection 6.4.9 features a detailed illustration of the workflow instrumentation necessary for (2).

Impact on the Architecture The PEP-WS receives web-service calls using ZXID, indicated by a small circle in the diagrams. The PEP-WS then sends a request to the PDP-Wf. The request contains the activity and workflow instance in question and a ZXID session ID referring to the identity information. If the request is granted, the PEP-WS forwards the call to the engine, which creates a new workflow instance. This new instance sends its ID to the PEP-WS, which registers the identity information for the start activity of that instance in the PIP-Wf.

6.4.6 Outgoing Web-Service Calls

For outgoing web-service calls made on behalf of users (Section 5.2.2.5), the WfMS first has to determine which service to call, then make the call. As such it has to include the correct credentials. Because we rely on unidirectional identifiers valid for one service provider only, the WfMS has to map the identifier of the user valid for itself to one valid for the provider of the service invoked.

ZXID provides the technical parts of the required functionality: Based on a ZXID session ID, ZXID can discover services of a given type available for the respective user, perform identity mapping, i. e. get credentials with identifiers valid for the service invoked, and perform the actual call according to the ID-WSF protocols. ID-WSF distinguishes between the sender identity (the user on whose behalf the call is made) and the invocation identity (the service provider actually performing the call). It also provides for the identification of intermediaries when service invocations are chained; see Sections 4.3 and 7.3 of [Hir06b]. The invoked service can perform access control according to its own policies, taking into account the invocation identity and the intermediaries that were part of the call chain.

6.4.6.1 Determining the Identity

Problem The WfMS first has to determine in whose name it has to perform the call, i. e. which identity to use.

Alternatives There are two main alternatives for solving this problem: (1) When a workflow instance performs an activity a_r receiving an incoming web service call, a handle representing the acquired identity information is provided as part of the message and can be stored in a (BPEL) variable. The handle can then be used in an activity a_i invoking a web service, causing this call to be performed on behalf of of the same user. (2) For each activity invoking a web service (a_i for short), a policy has to specify another activity (a_r for short) – either a human task or an activity that receives a web-service call. The WfMS then uses identity information from a_r to perform a_i. The difference to to Alternative 1 is that the identity information is not selected by explicit workflow logic, but through the evaluation of a dedicated policy.

Discussion While (1) is more flexible, (2) is in line with D6. It does not require changing existing workflow definitions; a small additional configuration is sufficient.

Conclusion Based on this argument, we choose Alternative (2).

Impact on the Architecture There has to be a component to evaluate the additional configuration. We assign this task to the PDP-Wf as it involves policy evaluation, even though the decision is not for access control purposes. The PEP-WS asks the PDP-Wf which credentials to use for a_i. The PDP-Wf determines a_r and looks up the identity information belonging to the last execution of a_r in the PIP-Wf.

Example 21 (Web-service call using identity information)*: In our earlier example, a_r is a human task in which the user confirms the flight details. a_i is an activity that invokes a flight booking service. a_i is executed shortly after a_r. The user knows that performing a_r will trigger the booking, so they agree to the use of their credentials to perform the call to the booking service.*

6.4.6.2 Performing Service Discovery

Problem When the PEP-WS knows which identity information to use, it can perform service discovery. To select a service, it has to take user preferences into account. We assume that these preferences have been coded explicitly as personalised trust policies.

Alternatives We see several alternatives for dealing with these policies. (1) The Trust PDP and the component that stores the policies (Trust Policy Store) are separate components. The Trust PDP retrieves the user's policy from the policy store based on the user's identity. (2) Like Alternative 1, but it is the component interested in the trust policy decision that retrieves the user's policy from the policy store and provides it to the Trust PDP as part of the request. (3) The Trust PDP has an internal trust policy store on its own. Note that in a distributed setting, users should be able to choose a Trust PDP, as well as the Trust Policy Store. Thus, which Trust PDP is used and from which Trust Policy Store the policy is retrieved depends on the user.

Discussion Alternative 2 discloses the trust policy to the WfMS, the other methods do not. Alternatives 1 and 2 would require trust policies to be expressed in a common policy language. This is not the case for Alternative 3, which is therefore more interoperable, as there is currently no common trust-policy language in widespread use. An argument in favor of one central Trust PDP would be that it able to compute a trust ranking for all services available.

Conclusion Our current implementation is similar to (2). The reason is that we are using an existing Trust PDP that supports dynamic policies passed at runtime but does not store user policies itself. At the moment, we are only using one central Trust PDP. This means that users cannot choose between several. One advantage of this is that a central Trust PDP has access to more user feedback and can thus deliver more accurate trust rankings.

Algorithm 2 PEP-WS performing a call to an identity web service

Require: activity, instanceId, svctype, payload
  $identity \leftarrow PDPWf.getIdentity(activity, instanceId)$
  $services \leftarrow Discovery.getServices(svctype, identity)$
  $maxScore \leftarrow -\infty$
  $bestService \leftarrow null$
  $policy \leftarrow PolicyStore.getPolicy(identity)$
  for all *service* in services **do**
    $score \leftarrow TrustPDP.getScore(service)$
    if $score > maxScore$ **then**
      $maxScore \leftarrow score$
      $bestService \leftarrow service$
  if $maxScore < 0$ **then**
    return *error*
  $result \leftarrow bestService.call(payload)$

Impact on the Architecture This design decision leads to the inclusion of two external components in our architecture. The PEP-WS retrieves user policies from the *Trust Policy Store* and then retrieves trust scores from the *Trust PDP*. See Algorithm 2.

6.4.7 Providing Attributes to Workflow Instances

There are two problems to be solved with respect workflow instances accessing identity attributes. The first problem is how the interface available to workflow instances should look like. This is addressed in Subsection 6.4.7.1. The second problem is how users should be informed about which identity attributes may be used by a workflow instance and can give their consent. Subsection 6.4.7.2 addresses this question.

6.4.7.1 Interface to be used by workflow instances

Problem Workflows must be able to access user attributes to provide a personalised service (Section 5.2.2.3).

Alternatives There are different ways to accomplish this (cf. [Gör09] and Subsection 3.8.2). They include (1) extending the interfaces of the WfMS so that all attributes available are provided once the workflow instance receives a call or a human task is completed, or (2) letting workflow instances explicitly fetch the attributes they need.

Discussion The interface of (2) is simpler. In addition, (1) would require changing interfaces and transferring attribute values even when the workflow does not need them. In most cases, there is no need for workflow instances to access identifiers. In particular, one can ensure that two tasks are performed by the same user by specifying a binding-of-duty constraint, which is evaluated by the PDP-Wf. Moreover, when attributes are only needed by web services that the workflow instance invokes, it is not necessary that the workflow instance itself knows these attributes.

Conclusion We have chosen to implement Alternative (2). This option is very flexible and allows workflows to access identity information regardless of how it has entered the WfMS.

Impact on the Architecture We will provide an *Identity-Information Access Service* workflows can invoke in order to retrieve attribute values. Workflows have to specify their instance ID and the name of the activity that has aquired the identity information.

Example 22: In our travel booking example, we assume that executives are allowed to travel business class while other employees are limited to economy class. Thus, the workflow has to determine a user's professional status and pass the corresponding booking class information to the flight booking service. We assume that the identity provider passes on the professional status as an attribute. To do this, the workflow sends a request to the IdI Access service which includes the workflow-instance ID, the name of the activity ("Initiate booking"), and the name of the attribute ("`job_status`"). The IdI Access service replies with the value of the attribute as acquired by the specified activity. Depending on the value, which is either "executive" or "regular_employee", the workflow chooses the booking class and includes it in the call to the flight booking service.

6.4.7.2 Information of users about access of workflow instances to attributes

Problem As discussed in Subsection 5.2.2.1 and Subsection 5.2.2.3, users must be aware (and thus have to be informed) which identity attributes provided to the WfMS through SSO will be (potentially) accessible by workflow instances. At the time SSO is performed, it is not yet known with which workflow instances the user will interact. Accordingly, the information has to be provided to the user at a later point.

Alternatives We see three fundamental alternatives regarding when and how often this information is provided to the user: (1) Information is provided once during the execution of each workflow instance. An opportunity for this is the first task that the user performs in the workflow instance. (2) Each time the user is about to perform a human task (belonging to a specific workflow instance), they are informed which attributes the workflow will (potentially) access. When the user continues to perform the task, this is interpreted as consent to access the attributes according to this notification. This approach allows access to be requested to more or fewer attributes depending on which execution path of the workflow is taken. (3) The user is only informed when the workflow instance tries to access identity attributes; these attributes are only transmitted to the workflow instance if the user gives their consent.

In the case of (1) or (2), a further problem is how to determine which attributes will potentially access. One possibility is an analysis of workflow definitions to determine which calls to the IdI Access service occur. Another possibility is a to manually specify which attributes the workflow will potentially access.

Discussion With (1) and (2), the notification is done when the user performs a human task. By contrast, (3) potentially results in several notifications not connected to user interactions that would occur anyway. Moreover, a synchronous interface for accessing attributes is sufficient with (1) and (2). This is not possible with (3) because users have to grant (or deny) consent before the attributes can be provided to the workflow. With (2) (and even more so with (1)), the user is informed about potential disclosure of attributes that perhaps will not occur at all. This is not an issue with (3). This also makes it necessary to have a specification as to which attributes the workflow instance will potentially access. It might be possible to automatically derive this specification from the workflow definition. However, this is a difficult task because BPEL allows complex control and data flow so that it is not obvious which calls exactly might be made to the IdI Access service.

Conclusion With Alternative 2, user interactions are efficient while it is still possible to have different execution paths or phases of a workflow with different attribute requirements. Moreover, implementation does not pose particular difficulties. Therefore, we choose this approach.

In addition, we choose to define the attribute requirements manually due to the conceptual and technical difficulty of deriving them automatically. Note that this does not preclude using an automatic approach later, as the handling of the specifications is the same regardless of how they have been created.

Impact on the Architecture Each workflow definition has to be accompanied by an attribute requirements policy. For each human task, it has to specify which identity attributes of the user performing that task will be potentially accessed by the workflow. The worklist handler must be able to access the policy applicable to each task being performed. It must compute the intersection of the user's attributes and the attribute requirements of the workflow and show the result to the user.

6.4.7.3 Enforcement of the attribute requirements policy

Problem A workflow instance must not access attributes beyond what is specified in the attribute requirements policy, because that would not be covered by the user's consent. The question is how this can be ensured.

Alternatives (1) The WfMS trusts that the workflow only accesses attributes as specified in the policy. This can be supported by respective audits. (2) The IdI Access service enforces the policy and denies requests for attributes not specified in the policy.

Discussion Alternative 2 requires some additional implementation effort. Alternative 1 poses security risks that can only partially be mitigated by audits; these audits also cause additional cost.

Conclusion We choose Alternative 2 because it provides better security at reasonable effort.

Impact on the Architecture In line with Subsection 6.4.4.1, the actual policy evaluation should be performed by the PDP-Wf. The IdI Access service has to request the PDP-Wf's decision before responding to requests for attributes.

6.4.8 Configuration

In the previous subsections, we suggested that the components of the system must first be configured to perform the functionality required. We will now summarise which configurations are needed and determine how to make them available to the WfMS components that need them.

In Section 5.2.2, we examined the necessary configurations for FIM functionality used in a WfMS. Earlier in this subsection, we determined the components responsible. The need for another kind of policy became apparent in Subsection 6.4.7.2. This resulted in the following types of configuration to be used by our envisioned WfMS:

Activity policy: Access control policies for activities, consisting of two parts: authorisation on the level of individual activities (*activity-level constraints*) and constraints involving multiple activities (*workflow-level constraints*). This separation into two parts suggests itself given that the two parts are independent and evaluation works quite differently for each of them. This is similar to the approach of Bertino et al. (cf. Section 3.5). – Note that users' trust policies are not part of the configuration of the WfMS. This is because the WfMS merely retrieves them from a policy store and forwards them to the Trust PDP. XACML policies are an obvious way of expressing activity-level constraints. These policies need to specify which subjects are allowed to perform which actions on which resources. XACML uses attributes to describe all of these. More explicitly, the resource is specified by using activity names as a resource attribute. Unlike in the work of Bertino et al., the policy is not limited to roles for describing subjects. Instead, an arbitrary combination of attributes can be used, allowing for more flexible policies. Solutions for workflow-level constraints include BPCL from the work of Bertino et al. already mentioned, or a simple set of BoD and SoD constraints each referring to two or more activities.

Example 23 (Two-part access control policy for activities)*: The first part, activity-level constraints, may take the following form. As XACML is very verbose, we state this part in natural language; translation into XACML is straightforward.*

- *For resources with* `activity_name = "Initiate Booking"` *or* `activity_name = "Choose Airline"` *and action "*`perform`*", require subject attribute* `employment_status = "regular"`*.*
- *For resources with* `activity_name = "Authorise Travel"` *and action "*`perform`*", require subject attribute* `job_status = "executive"`*.*

The second part, workflow-level constraints, is a set of BoD and SoD constraints:

- *BoD("Initiate Booking", "Choose Airline"): The employee initiating the travel booking workflow must confirm the details of their trip.*

- *SoD("Initiate Booking", "Authorise Travel"): No employee, not even an executive, may approve their own trip.*

Identity selection policy: For each activity performing an outgoing web-service call, the identity information to be used has to be given by specifying the activity that has acquired this information. This is just a set of pairs of activity names, such as {(Confirm Booking, Book Flight)}, which indicates that the identity information acquired by the "Confirm Booking" activity is to be used for the web-service call in the "Book Flight" activity.

Attribute requirements policy: As explained briefly in Subsection 6.4.7.2, this policy needs to list the attributes a workflow potentially requires with respect to the user performing each task in the workflow. A straightforward way to express this is by listing human tasks and a set of attributes for each one. Human tasks for which no attributes of the user performing them are required can be omitted. As an example, {(Request Booking, {`job_position`})} indicates that the workflow requires the `job_position` attribute (so that it can decide which booking classes are allowed according to the organisation's rules for business trips.

As shown in Figure 6.2, accompanying security configuration has to be created in addition to workflow definitions, e. g. as special annotations to workflow activities (cf. also Subsection 6.1.1). After a transformation into the policies defined in the previous paragraph, the workflow definitions alongside with these policies have to be deployed to the WfMS. This is done by a special tool triggered from the workflow modelling component. This tool has to make sure the user is allowed to deploy workflow definitions (according to a system-wide policy) and has administrative access to the components of the secure WfMS. The identity selection policy and the attribute requirements policy have to be deployed to the PDP-Wf, using the identifier of the workflow definition in the Workflow Engine. There are alternative ways of providing the activity policy to the stateless PDP: (1) The activity policy is deployed to the PDP-Wf, which provides it with every authorisation request. This is not possible with every PDP implementation and can be inefficient. (2) The part of the policy relating to individual activities (1) is deployed directly to the stateless PDP. Requests made by the PDP-Wf to the stateless PDP include the name of the workflow definition, so that the stateless PDP can determine the policy applicable. – Our current approach is based on Alternative 1. We propose switching to Alternative 2 for efficiency reasons.

6.4.9 Correlation of Incoming Web-service Calls

In Subsection 6.4.5.3, we raised the question of how the PEP-WS can determine the workflow instance and activity that an incoming web-service call is directed to. This is necessary so that the PEP-WS can perform authorisation before the message is actually processed by the workflow instance. We sketched a way to accomplish this, and we now describe it in more detail.

Consider a workflow with a `<receive>` activity r_1 that receives a message with payload P (Figure 6.4). For each such activity, the workflow definition is instrumented as follows:

- The interface of r_1 is extended so that the incoming message also contains a message ID m_1 created by the PEP-WS.

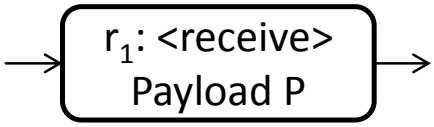


Figure 6.4: Original `<receive>` activity

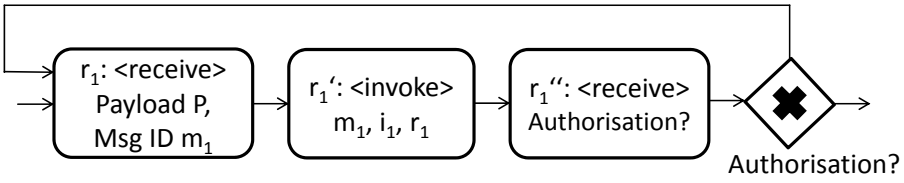


Figure 6.5: Fragment inserted for the `<receive>` activity

- After r_1, an `<invoke>` activity r_1' is inserted, which calls the PEP-WS to inform it of the workflow instance (i_1) and activity (r_1) that m_1 was directed to.
- Another `<receive>` activity r_1'' is inserted after r_1'. It retrieves the authorisation decision from the PEP-WS. If authorisation is granted, execution of the workflow instance continues normally. If not, execution of r_1 restarts.

Figure 6.5 shows the result of the instrumentation in a form similar to BPMN. The corresponding instrumentation in WS-BPEL is straightforward. Assume r_1 originally looks as follows:

```
<receive partnerLink="PL1" portType="PT1"
 operation="sendToPL1" variable="v1" name="r1" />
```

This is transformed into the following:

```
<sequence>
 <repeatUntil>
  <sequence>
   <receive partnerLink="PL1Ext" portType="PT1Ext"
    operation="sendToPL1" variable="v1ext" name="r1" />

   <assign><!-- Copy m1, i1 and r1 into
          a variable context --></assign>

   <invoke partnerLink="PEPWS"
    portType="PEPWSContextNotification"
    operation="informAboutContext"
    inputVariable="context" />

   <receive partnerlink="PEPWS"
    portType="PEPWSContextNotificationCallback"
```

```
    operation="sendAuthorisationDecision"
    variable="authorisationDecision" />
  </sequence>
  <condition>
   <!-- authorisation granted -->
  </condition>
 </repeatUntil>

 <assign>
  <!-- copy payload part of v1ext into v1 -->
 </assign>
</sequence>
```

6.4.10 Assessment of Design Requirements

In Subsection 5.2.1, we introduced a number of design requirements. We can group these into privacy-related (D1 and D2) and architecture-related (D3 to D6). We will now assess our architecture with respect to these design requirements by means of plausibility arguments.

D1 requires preventing workflow instances from combining identity information. We allow workflow instances to access *any* identity attribute of users that could serve as a quasi-identifier. As such, workflow definitions must be validated before deployment to ensure they adhere to the respective privacy policy. In particular, workflows may only correlate identity information from different sources when this is necessary to fulfil the workflow's purpose and is specified in the workflow's policy.

According to D2, identity information should be released only with the user's consent. In light of this requirement, it is necessary to distinguish which entity is responsible for requesting the user's consent. When an external caller invokes a web-service interface of the workflow or when the user logs in via SSO through an identity provider, the transfer of identity information is initiated by a component that is not part of the WfMS. We see this component (the caller or identity provider respectively) as responsible for ensuring that the user has consented to this transfer. The WfMS itself is responsible when it provides identity information to workflow instances or performs calls to external web services that include identity information. Identity information included in web-service calls is directed to a specific workflow instance, to which the WfMS provides it without further checks. Identity information acquired through SSO is only provided to a workflow instance when the user performs a task in the instance. The attribute requirements policy (which is enforced automatically) states which attributes the workflow instance will potentially access, and the user is informed accordingly before completing a task. This means attributes are only disclosed to workflow instances with the user's informed consent. When the WfMS invokes a web service, it checks that the service is trustworthy according to the user's policy (Subsection 6.4.6.2). The identity provider issues an assertion containing the user's identity information specifically for the service invoked. This means the identity provider can (and should) control which information to release.

D3 postulates abstraction from the actual technologies used. Firstly, we achieve this by building the architecture around generic concepts of identity

management and access control: The fundamental concepts used, namely identifiers, attributes, and the transfer of identity information in service calls and user interactions, are independent from any particular specification such as SAML. Secondly, the architecture uses a dedicated software layer to encapsulate the details of the technology actually used (Subsection 6.4.2). This layer has a sufficiently generic interface based on technology-independent concepts. It provides high-level interfaces for acquiring identity information through web-service calls and user interactions, accessing it using generic concepts such as *identifier* and *attribute*, and outputting it for access control or web-service calls.

D4 requires a standards-based architecture, and D5 aims at clearly specifying the component responsible for a certain piece of functionality. The standards relevant for our architecture are the WfMC reference model for workflow management functionality, and the XACML reference architecture for its access control functionality. Following existing standard architectures is also one important step towards a clear assignment of functionality to components. Following the WfMC reference model ensures that the functionality needed is implemented for all interfaces of the WfMS. In line with the XACML reference architecture, the tasks of policy enforcement, policy decision and storing information needed for this are assigned to dedicated components (Subsections 6.4.4.1 and 6.4.5.1). Our architecture specification also defines whether shared or distinct components are responsible for similar functionality with respect to human tasks and web services, and specifies how these interact (Subsections 6.4.5.2 and 6.4.5.3).

According to D6, declarative means of configuration should be preferred over explicit control of FIM functionality by individual workflows. The architecture employs declarative policies for access control and determining the identity to use in web-service calls (Subsection 6.4.8). These policies are independent from the workflow definition itself, so can be easily adapted when more powerful specification mechanisms become available. When workflows need attribute values for application-specific purposes, they must fetch them explicitly. We conclude that the WfMS uses declarative policies wherever possible in an application-dependent way.

In summary, the WfMS fulfils the architecture-related design requirements. We support the privacy-related requirements with respect to workflow-specific functionality. In addition, these requirements need to be supported by identity management technology in general.

6.4.11 Sample message flow and user interaction in the architecture

In the following, we provide a case study based on the travel booking example, demonstrating how the corresponding security specification is evaluated during workflow execution.

The example workflow definition, dubbed P, is a linear sequence of four activities. a_1 (*Initiate booking*) receives incoming (identity) web service calls that start the workflow. The interface comprises a destination, travel dates, and a reason as input. a_2 (*Authorise travel*) is a human task which displays destination, travel dates, reason, and the name of the traveller, plus asks for authorisation. a_3 (*Choose airline*) is a human task for the traveler. a_4 (*Book flight*) is a call to a web service provided by a travel agency.

This workflow definition is accompanied by the following security configuration: (1) Activity level authorisation: a_1 and a_3 require an attribute `employment_status` with the value `regular`. a_2 requires an attribute `position` with the value `manager`. (2) Workflow-level authorisation constraints: $\{\text{BoD}(a_1, a_3), \text{SoD}(a_1, a_2)\}$ (3) Identity selection for the outgoing WS call: $\{(a_3, a_4)\}$. – the WfMS executes a_4 in the name of the user who performed a_3.

We now list the steps in a sample execution of P, together with the component interactions they cause:

- John, a regular employee, initiates a WS call to the interface of a_1 to start the booking workflow for a trip to London on June 20–22 to visit a trade fair. The PEP-WS intercepts this call and uses the ID-WSF library to extract the identity information (IdI). Since a_1 is the initial activity of P, the PEP-WS creates a new workflow instance ID, P_1. It then asks the PDP-Wf for authorisation. The PDP-Wf in turn sees from the policy that a_1 is subject to a BoD as well as a SoD constraint. It looks up the IdI for a_2 and a_3 in the PIP-Wf. The result is that these activities have not yet been performed, meaning the constraints cannot be violated. The PDP-Wf then calls the stateless PDP, passing on John's IdI. As John is a regular employee, authorisation is granted. The PDP-Wf forwards this decision to the PEP-WS, which then calls a_1 in the Workflow Engine, passing on the new ID P_1, and stores the connection of John's IdI to a_1 in the PIP-Wf.

- The Workflow Engine starts the execution of a_2, sending a request to the PEP-HT. The PEP-HT assigns the ID h_1 to the new human task instance and registers this ID in the PIP-Wf together with P_1 and a_2. It then forwards the request to the worklist handler. Bob, a manager in John's department, then logs into the worklist handler via SSO, using the company's IdP. The ID-WSF library handles the SSO process and stores Bob's IdI. For all available tasks, including h_1, the worklist handler requests an authorisation decision from the PEP-HT. The PEP-HT determines that h_1 was created by a_2 in P_1 by asking the PIP-Wf. The PEP-HT then sends an authorisation request containing Bob's IdI, the ID of the activity (a_2), and the ID of the workflow instance (P_1), to the PDP-Wf. The PDP-Wf's policy states that a SoD conflict with a_1 exists. It looks up the IdI for a_1 in the PIP-Wf and determines that a_1 was performed by a different identity, namely John. It then requests authorisation from the stateless PDP. This is granted because Bob is an executive. The worklist handler receives the PDP-Wf's decision via the PEP-HT and shows h_1 in Bob's worklist. The same steps are executed for all other human tasks currently registered in the worklist handler; if Bob is (currently) allowed to perform them, they are shown in his worklist as well. Bob chooses h_1 and completes it, authorising John's trip. Because new authorisation constraints might apply due to activities executed in parallel, the worklist handler requests authorization again, and it is granted the same way as before. The PEP-HT registers h_1 as completed, with a pointer to Bob's PII. The worklist handler sends the completed task (Bob's decision) to the PEP-HT. The PEP-HT determines

the callback address for h_1 from the PIP and forwards the result to the Workflow Engine.

- The Workflow Engine starts the execution of a_3, sending a request to the PEP-HT. The PEP-HT assigns the ID h_2 for the new human task instance and registers it in the PIP-Wf together with P_1 and a_3. It then forwards the request to the worklist handler. John then logs into the worklist handler via SSO, using the company's IdP. The ID-WSF library handles the SSO process and stores John's IdI. For all available tasks, including h_2, the worklist handler requests an authorisation decision from the PEP-HT. The PEP-HT determines that h_2 was created by a_3 in P_1 by asking the PIP-Wf. It then sends an authorisation request containing John's IdI, a_3, and P_1, to the PDP-Wf. The PDP-Wf's policy states that a BoD relation with a_1 exists. It looks up the IdI for a_1 in the PIP-Wf and determines that a_1 was performed by the same identity. It then requests authorisation from the stateless PDP. This is granted just as for a_1. The worklist handler receives the PDP-Wf's decision via the PEP-HT and shows h_2 in John's worklist. Again, the same steps are executed for all other human tasks currently registered in the worklist handler; if John is (currently) allowed to perform them, they are shown in his worklist as well. John chooses h_2 and completes it, choosing Universum Airways. The worklist handler requests authorisation again, which is granted the same way as before. The PEP-HT registers h_2 as completed, with a pointer to (the new copy of) John's PII. The worklist handler sends the completed task to the PEP-HT. The PEP-HT determines the callback address for h_2 from the PIP and forwards the result to the Workflow Engine.

- The Workflow Engine starts execution of a_3, a request to book a flight to London with Universum Airways around June 22-24. Its message to the PEP-WS includes a_4, P_1 and the service type, i.e. "flight booking". The PEP-WS asks the PDP-Wf which identity to use for the call. The policy states that the identity from a_3 should be used, so the PDP-Wf looks up who has performed a_3 in P_1 in the PIP-Wf. It sends the result, a pointer to John's IdI, to the PEP-WS. The PEP-WS now performs service discovery using the ID-WSF library and discovers two flight booking services, b_1 and b_2. It then retrieves John's trust policy from his trust policy store. The PEP-WS calls the Trust PDP with this policy to retrieve trust scores for b_1 and b_2. According to his policy, John trusts b_2, but not b_1. The PEP-WS accordingly calls b_2, using John's credentials. The ID-WSF library performs identity mapping automatically through the *Identity Mapping Service*.

One can see that a simple workflow accompanied by a simple corresponding policy led to many actions performed by the WfMS in the background. This would be very tedious for workflow designers to specify explicitly. Our WfMS, by contrast, performs the required actions automatically. This facilitates the use of FIM for workflows and improves reliability with respect to identity management.

6.5 Summary

In this chapter, we have developed architecture extensions in two steps, leading to a secure and identity-aware workflow management system. We have also investigated how this system can be configured and which kinds of workflow context it can adapt to. While we have not addressed all requirements, the system is flexible enough to cope with the remaining ones without major changes, as we will show in Section 8.2.

Chapter 7

Implementation

In the previous chapter, we described the architecture of our secure workflow management system, i. e. its components and their relationship. Now we will describe the technical aspects needed to actually implement the components themselves and the communication links between them[1].

The remainder of this chapter is structured as follows:

- We begin with an overview of the components defined in both variants of the architecture, briefly explain their functionality again, and analyse which components from both variants correspond to each other. We also hint at possible discovery methods that would allow the components to find each other.
- We decide which library to use for handling the technical details of SAML and ID-WSF.
- We determine which data each component uses, either directly or by referencing it. We define the data formats used for literal data, and the identifier schemes used for references to data.
- Based on several scenarios covering the identity management functionality described in the previous chapter, we analyse the interaction of components. We consider which messages components exchange in these scenarios, and define XML data formats for these messages. This provides a sound definition of the communication.
- We explain the process of deploying security configurations for workflow definitions, and of deploying the workflow definitions themselves. We elaborate on the necessary transformations of workflow definitions.
- We show how to address performance bottlenecks in the architecture and its implementation based on one example.
- We explain in detail how to achieve the necessary transformations of workflow definitions.
- We briefly examine the use of legacy software components.

[1] Parts of this chapter are based on the author's contribution to [TAS11], a public project deliverable accepted by the European Commission.

Secure WfMS	WfMS/FIM	Remark
Workflow Engine	Workflow Engine	*no differences*
PDP-Wf	PDP-Wf	*The latter is specified in more detail.*
PIP	PIP-Wf	*The latter additionally stores identity attributes.*
PAP-Wf	–	*In the latter architecture, PAP functionality is included in the PDP-Wf.*
Context Handler	IdI Access	*The latter has a narrower scope.*
PEP-WS	PEP-WS	
Key Store	–	*Considered an implementation issue in the latter version, part of ZXID.*
Trust PDP	Trust PDP	
–	Trust Policy Store	*Identity-based service*
Service Discovery	IdP	*Described as part of the IdP, following ID-WSF specifications.*
UInvolve Handler	–	
Async UI	–	
PEP-HT	PEP-HT	
Worklist Handler	Worklist Handler	
–	IdP	*Also handles service discovery.*
–	PDP	*Handles non-workflow-specific decisions.*

Table 7.1: Components described in both versions of the architecture.

7.1 Component Overview

Table 7.1 lists the components described in the basic architecture for a secure WfMS and the version of the architecture that integrates federated identity management. The main differences are as follows: Some names have changed slightly. The handling of service discovery and key management is different due to the introduction of ZXID and identity management. Security-specific and asynchronous user interactions are not addressed in the latter architecture.

Next, we consider how other components must be able to discover these components:

- **Workflow Engine:**
  - Replies to outgoing requests are either made in the same connection or to an address specified in the original request. This address must allow identification of the respective workflow instance and partner link.
  - Requests to start new workflow instances must be directed to specific preconfigured endpoints. These endpoints must be published, e. g. in a service directory.

- **Workflow PDP:** This component is needed only by other components of our architecture. Thus, its address can be hardcoded in the configuration of these components.
- The same holds for the **PIP-Wf**, the IdI Access component, the **UInvolve Handler** and the **PDP**.
- The **Context Handler** provides status information about workflow instances to external components. It thus has to notify them that relevant information is available. For example, a user's control centre should know about the workflow instances the user is involved in. This notification may involve an additional intermediary. Once the initial notification has been made, the control centre can retrieve additional information.
- **PEP-WS**:
  - First, the PEP-WS handles requests from the Workflow Engine to external web-services. This is achieved by instrumenting the workflow to call the PEP-WS instead. The PEP-WS's address can be preconfigured.
  - Second, it also handles web-service messages sent *to* the engine. Because SOAP is based on HTTP, one can simply redirect HTTP connection requests to the engine to the PEP-WS. The HTTP headers still contain the original address, so the PEP-WS can forward them after applying authorisation and protocol translations.
- The **Trust PDP** is part of the overall security infrastructure, usually including more than one workflow engine and associated security components. However, its address can still be preconfigured in the PEP-WS.
- The responsible **IdP** is determined based on the user who authenticates to the system. A trust relationship is established before the system accepts authentication provided by a given IdP. A session is always established when a user logs in using one of the preconfigured IdPs.
- **Service Discovery** is performed by a discovery service that may be different for each user. The responsible service is determined when a user's IdP provides authentication to the system.
- In our architecture, there is only one **Worklist Handler** per workflow management system. It is therefore preconfigured in the PEP-HT. It might be desirable to have user-specific worklist handlers so that users can see task from different workflow management systems in one place. This would require an allocation mechanism, as potentially distributing every task to every user would be infeasible. The worklist handler for users to whom a task has been allocated can then be determined using service discovery.
- **Async UI** is not covered in detail. However, the same considerations as for the Worklist Handler apply.

Table 7.2 and Table 7.3 list the types of data handled by each component and the identifiers used by it. Based on this overview, we will then select suitable data formats and identifier schemes.

Component	Type of data used:	Identifiers used for:
Wf Engine	Workflow definitions Runtime information about workflow instances	Workflow definitions Workflow instances Workflow activities
PDP-Wf	Policies Identity information	Workflow definitions Workflow instances Workflow activities Blocks of identity information Users
PIP(-Wf)	Identity information obtained Status of human tasks	Blocks of identity information Users Workflow definitions Workflow instances Workflow activities
Context handler	Status of workflow instances and activities	Workflow definitions Workflow instances Workflow activities
IdI Access	Attributes	Workflow definitions Workflow instances Workflow activities Blocks of identity information Attributes
PEP-WS	Trust levels of web services SOAP messages to and from web services	Web-service endpoints Web-service types Workflow definitions Workflow instances Workflow activities
Key Store	Cryptographic keys of web services	Web-service endpoints
Trust PDP	Ratings (and other sources for computing trust scores)	Web-service endpoints Web-service providers Users
Trust Policy Store	Trust policies	Users
IdP/Service Discovery	Registration of services Authentication information for users	Web-service endpoints Service types Users

Table 7.2: Data and identifiers used by the components (part 1).

Component	Type of data used:	Identifiers used for:
UInvolve Handler *and* Async UI	Security configuration	Workflow definitions Workflow instances Workflow activities Users Human tasks Web-service endpoints *Other objects referred to in security configuration*
PEP-HT	Status information about active human tasks	Workflow definitions Workflow instances Workflow activities Instances of human tasks Blocks of identity information
Worklist Handler	Definitions of human tasks Active human tasks (Parts of) attribute requirements policies	Definitions of human tasks Instances of human tasks attributes
PDP	Policies Identity information	

Table 7.3: Data and identifiers used by the components (part 2).

7.2 Library for SAML and ID-WSF

As decided in Subsection 6.4.2, a library should be used to handle the SAML and ID-WSF protocols. Several libraries that provide the desired functionality are available. ZXID[2] is a library that implements the ID-WSF protocols. Other possibilities are the Liberty Open Source Toolkit[3], another implementation of ID-WSF, and OpenSAML[4], which implements only SAML itself, not the ID-WSF that is built on top of it.

In comparison with SAML, ID-WSF specifies additional details facilitating interoperability. This makes it advantageous to choose a library supporting it. ZXID persists identity information as *ZXID sessions* and lets these be addressed using so-called *ZXID session IDs*. It can perform SSO and extract identity information from ID-WSF-compliant web-service calls. Both functions lead to the creation of a ZXID session, which stays available until it is explicitly deleted. ZXID can make outgoing ID-WSF calls and request authorisation decisions using identity information from a specified ZXID session. Finally, ZXID provides an interface to access attributes and identifiers. By sharing ZXID session IDs, components running on the same machine can use ZXID to access identity information.

[2] `http://www.zxid.org/`
[3] `http://www.cahillfamily.com/OpenSource/`
[4] `https://wiki.shibboleth.net/confluence/display/OpenSAML/Home`

ZXID is used throughout the architecture. First, several components use ZXID for external communication by means of a special protocol supported by ZXID. Second, these components access identity information (identifiers or attributes) already stored in a ZXID session.

7.3 Data and Identifier Formats

The previous section lists the data and identifier types used by the architecture components. Here, we describe the formats these take.

- **Workflow definitions** follow the WS-BPEL specification [JE07] (Subsection 2.3.2).
- The **runtime information about workflow instances** needed by the workflow engine is also given in the WS-BPEL specification. However, not all of this information is required by other components. Only the information on whether a given activity in a given workflow instance is currently active is needed.
- Possible elements for **identity information** are described in Subsection 2.4.5.1. A block of identity information can be expressed by a SAML assertion. On a more fine-grained level, there are pairs of attribute types and attribute values. We will discuss identifier formats for attribute types below. Attribute values can be expressed as simple strings.
- **Access control policies** specify who is allowed to execute which activities. They contain rules based on the attributes of the respective user and on the activity in question. This can be achieved using XACML policies (Subsection 2.4.3.1). How user attributes should be referred to is specified in the SAML 2.0 Profile for XACML (Subsection 2.4.5.3). The name of the activity can be specified as a resource attribute. For each workflow definition, a separate XACML policy is specified. In addition, history-based constraints (separation of duty, binding of duty) must be be specified in a non-standard format.
- The **status of human tasks** can be expressed straightforwardly: For each human task instance, an identifier of the instance and of the human task definition must be stored. If the human task has been claimed by a user, a reference to their identity information must also be stored. Section B.2 features a detailed explanation of possible states of human tasks.
- **Trust levels** of web services are simply numeric values.
- Messages sent to and received from web services are in SOAP format (Subsection 2.2.2).
- The cryptographic keys of web services can be embedded as part of the `SecurityContext` element of an ID-WSF endpoint reference (cf. Section 2.3.3 of [CC07]).

Type of object	Context	Format
Workflow definitions	WfMS	String
Workflow instances	WfMS	Number
Workflow activities	Workflow definition	String
Block of identity information	WfMS	ZXID session ID
Users	WfMS[5]	String
Attribute types	Global	LDAP object ID
Web-service type	Global	URN
Web-service instance	Global	Endpoint URL
Human task definition	WfMS	String
Human task instance	WfMS, Worklist Handler	Number

Table 7.4: Identifier formats.

- User ratings, expressed as tuples of a persistent user ID, a service ID and a value, can act as a simple **basis for computing trust levels**. **Trust policies** are user-specific. Both types of information are out of scope here. [BEdH+10] provides a classification of possible trust policies.
- **Service registrations** consist of a service type and a service endpoint.
- The identity provider must be able to authenticate registered users. Passwords are a simple example of this type of **authentication information**.
- Section B.1 proposes a format for **definitions of human tasks**.

In addition, several types of objects must be uniquely identifiable. Table 7.4 lists the context in which these have to be unique, and the format used.

7.4 Interaction of Components

Up to now, we have looked at the implementation of individual components and the data formats used. As the next step, we will now analyse the interactions between components. To this end, we will look at different scenarios covering all interactions between components. These are as follows:

- A workflow instance creates a human task instance.
- A user views the list of their tasks.
- A user performs a human task.
- A user tries to perform two tasks, and a separation-of-duty constraint applies.
- The workflow invokes a web service, including the computation of trust rankings and service selection.
- An incoming web-service call starts a new workflow instance.

[5]For different service providers, different identifiers (pseudonyms) are used. Automatic translation is performed in a privacy-preserving way.

- A workflow definition and accompanying security policies and human task definitions are deployed.

We will now explain these interactions in more detail, define interfaces and review how data formats and identifiers are used in these interactions.

7.4.1 A workflow instance creates a human task instance

This scenario involves the workflow as well as the PEP-HT, which handles the creation of the task instance, the PIP-Wf, which stores information about the newly created task instance, and the worklist handler.

Schematically, the interaction is as follows:

1. The workflow sends a request to create a human task instance. Instead of sending it directly to the worklist handler, the workflow sends it to the PEP-HT, which acts as a proxy. This request contains information about the context of the task, e.g. the workflow instance.

2. The PEP-HT creates a new, unique task instance identifier. It stores the identifier and information about the task instance in the PIP-Wf so that it is available later for the evaluation of policy decision requests.

3. The PEP-HT then forwards the request to the worklist handler, including the task instance identifier.

Extending the data and identifier formats defined above, we will now discuss the interface for creating an instance of a human task:

Creating tasks requires only three fixed parameters:

- an ID of the task instance, created by the PEP-HT,

- the ID of the task definition (which must have been deployed previously), and

- a callback URL to which the notification of the completion of the task will be sent. The format of the callback message will be defined in a separate section below.

The other parameters depend on the input parameters defined in the task definition. While it would be possible to define an interface (as a WSDL) that works for all task definitions, such an interface description would not be very useful for creating stubs for calling the method or validating invocations. Therefore, we will dynamically create a WSDL interface description for each task definition.

For the task definition given above, the body of a SOAP message would look as follows (note that we use the document/literal wrapped convention for SOAP calls):

```
<createTask xmlns="urn:tas3:examples:taskdef1:createRequest">
   <!-- ":createRequest" added to the ID of the task definition -->
 <taskid>taskid-0815</taskid>
 <taskdef>urn:tas3:examples:taskdef1</taskdef>
   <!-- This may seem redundant, but makes it possible to determine
```

```
     the applicable task definition without splitting the string in
     the namespace declaration. -->
 <callback>http://example.org/axis2/taskdef1Callback/</callback>
 <taskInput>
  <number_x>5</number_x>
  <number_y>6</number_y>
  <x_or_y>same</x_or_y><!-- the preselected choice -->
  <correct>false</correct><!-- the checkbox will be
                               unchecked initially. -->
  <flavour>
   <choice value="vanilla">Vanilla</choice>
   <choice value="choco">Chocolate</choice>
   <choice value="forest">Forest Fruits</choice>
  </flavour>
  <privacy_policy caption="Privacy policy"
       url="http://example.org/privacy_policy" />
 <taskInput>
</createTask>
```

A generic schema (without the parts specific to each task definition) for this message body looks as follows:

```
<!-- In each generated schema for a specific task definition, a
     different targetNamespace will be used. -->
<schema targetNamespace="urn:tas3:examples:taskdef1:createRequest"
        xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="createTask">
    <complexType>
      <sequence>
        <element name="taskid" type="string"/>
        <element name="taskdef" type="string"/>
        <element name="callback" type="string"/>
        <element name="taskInput">
          <!-- to be replaced (as explained below) by
               a more specific definition for the
               particular task definition. -->
          <sequence>
            <any namespace="##any" minOccurs="0" />
          </sequence>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

As indicated, we need to fill in the type definition of the taskInput element. For each input parameter of the task, we need to add one element, resulting in a sequence. The schema elements correspond to the possible parameters as follows:

- For text inputs:

```
<element name="id of the text input" type="string" />}
```

- For boolean inputs (checkboxes):

```
<element name="id of the boolean input" type="boolean" />}
```

- For a dynamic list of choices:

```
<element name="id of the dynamic_choice input" >
 <complexType>
  <sequence>
   <element name="choice" minOccurs="0"
                          maxOccurs="unbounded">
    <complexType>
     <sequence>
      <element name="caption" type="string">
      <element name="id" type="string">
     </sequence>
    </complexType>
   </element>
  </sequence>
 </complexType>
</element>
```

- For a static list of choices:

```
<element name="id of the static_choice input" type="string" />}
```

- For a URL:

```
<element name="id of the URL form element" >
 <complexType>
  <sequence>
   <element name="caption" type="string">
   <element name="url" type="string">
  </sequence>
 </complexType>
</element>
```

Given these definitions, the schema definition for our sample task definition looks as follows:

```
<schema targetNamespace="urn:tas3:examples:taskdef1:createRequest"
        xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="createTask">
    <complexType>
      <sequence>
        <element name="taskid" type="string"/>
        <element name="taskdef" type="string"/>
        <element name="callback" type="string"/>
        <element name="taskInput">
         <complexType>
          <sequence>
```

```
            <element name="number_x" type="string" />
            <element name="number_y" type="string" />
            <element name="x_or_y" type="string" />
            <element name="correct" type="boolean" />
            <element name="flavour">
             <complexType>
              <sequence>
               <element name="choice" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                 <sequence>
                  <element name="caption" type="string">
                  <element name="id" type="string">
                 </sequence>
                </complexType>
               </element>
              </sequence>
             </complexType>
            </element>
            <element name="privacy_policy">
             <complexType>
              <sequence>
               <element name="caption" type="string">
               <element name="url" type="string">
              </sequence>
             </complexType>
            </element>
           </sequence>
          </complexType>
         </element>
       </sequence>
     </complexType>
  </element>
</schema>
```

The task creation component will be provided as a servlet and has to offer two different functionalities. First, it offers a WSDL describing the task creation interface. Second, it provides a (generic) interface that accepts such requests. It is sufficient to provide one endpoint here (independent of the task to be created), as requests can be dispatched based on the namespace and the `taskdef` element.

An example of a full WSDL is given below. The URL where the WSDL can be retrieved depends on the location where the servlet is deployed.

```
<wsdl:definitions
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="urn:tas3:examples:taskdef1:createRequest"
   targetNamespace="urn:tas3:examples:taskdef1:createRequest" >
 <wsdl:types>
  <schema targetNamespace="urn:tas3:examples:taskdef1:createRequest"
```

```
        xmlns="http://www.w3.org/2001/XMLSchema">
   <complexType name="createTaskRequestType">
    <!-- Insert the sequence element that appears as the child of
           the complexType element in the definition of the createTask
           element in the schema above. -->
   </complexType>
   <element name="createTask" type="tns:createTaskRequestType" />
   <element name="createTaskResponse">
    <complexType/>
   </element>
  </schema>
 </wsdl:types>

 <wsdl:message name="createTaskRequest">
   <wsdl:part name="root" element="tns:createTask"/>
 </wsdl:message>

 <wsdl:message name="empty">
    <wsdl:part name="parameters" element="tns:createTaskResponse"/>
</wsdl:message>

 <wsdl:portType name="createTaskPT">
  <wsdl:operation name="createTask">
   <wsdl:input message="tns:createTaskRequest"/>
   <wsdl:output message="tns:empty"/>
  </wsdl:operation>
 </wsdl:portType>

 <wsdl:binding name="createTaskSoapBinding" type="tns:createTaskPT">
  <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="createTask">
   <wsdl:input><soap:body use="literal"/></wsdl:input>
   <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>
 </wsdl:binding>

 <wsdl:service name="CreateTaskSoapService">
  <wsdl:port name="createTaskPort" binding="tns:createTaskSoapBinding">
   <soap:address
     location="http://example.org/CreateTaskServlet/createTask"/>
     <!-- determined by the implementation -->
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

This interface is provided by the worklist handler and used by the PEP-HT. There is another interface provided by the PEP-HT and used by the workflow instance. This interface is essentially the same. However, it has to be extended with context information needed for authorisation. We will discuss this issue

below when we address the necessary workflow transformations.

The worklist handler can store the information using two database tables described in Section B.3.

To register the task instance in the PIP-Wf, the following information is necessary:

- The ID of the workflow instance (which can also be used to deduce the ID of the workflow model),
- the new task instance ID created by the PEP-HT,
- the information that the task has not yet been performed and no user has claimed it, and
- the callback URL that will be used to inform the workflow when the task has been completed.

7.4.2 A user views the list of their tasks

In this scenario, we assume that a human task instance has been properly created and is registered in the PIP-Wf and the worklist handler. Schematically, the following steps occur for each task registered in the worklist handler when a user logs in and views their task list:

1. For each available instance of a human task, the worklist handler sends its ID and a reference to the user's identity information to the PEP-HT and asks for authorisation.
2. The PEP-HT retrieves information about the task instance from the PIP-Wf.
3. The PEP-HT sends a policy decision request with all of this information to the PDP-Wf.
4. The PDP-Wf reads the policy applicable to the workflow model in question from its policy store.
5. Using the policy, it determines other task instances relevant to the current decision request. Other task instances can be relevant because of binding-of-duty or separation-of-duty constraints.
6. The PDP-Wf determines who has executed these task instances by invoking the PIP-Wf.
7. The PIP-Wf sends a reply with the requested assignments.
8. It then evaluates the policy and answers the request. Part of this conceptual step is actually performed by a separate, stateless PDP. Accordingly, this step can be split up as follows:
   a) First, the PDP-Wf checks whether constraints are fulfilled. If not, it sends a *Deny* response to the PEP-HT's request.

   b) If all constraints are met, the PDP-Wf checks whether the user is allowed to perform the task based on their attributes. To do so, it contacts a general-purpose PDP, passing the policy applicable to the task in question and the user's attributes in SAML format.
   c) The (general-purpose) PDP evaluates the policy and returns a decision to the PDP-Wf.
   d) The PDP-Wf returns the overall decision to the PEP-HT.

9. The PEP-HT forwards the decision to the worklist handler, which, depending on the decision, includes the task in the worklist shown to the user or not.

The worklist handler's user interface consists of two views:

- Task list: A page that shows the tasks available to the current user in a table, showing the title of the task and whether the task is already claimed by the current user.
- Task display: This screen shows a task with a caption, text explanation and form elements according to the task definition. The values of the form elements are initially set according to the task creation request. The user can either save a task, causing the entered data to be persisted, or complete it. In the latter case, the data the user entered will be sent back to the workflow.

The following interfaces are used in the steps described above:

- The PEP-HT handles authorisation requests for human task instances.
  - Input parameters: ZXID session ID, task instance ID
  - Result: Boolean value
- The PIP-Wf provides information about a human task instance.
  - Input parameter: Task instance ID
  - Result: Workflow model ID, workflow instance ID, workflow task ID
- The PDP-Wf handles authorisation requests for instances of workflow tasks.
  - Input parameters: Workflow model ID, workflow instance ID, task ID, ZXID session ID
  - Result: Boolean value
- The PIP-Wf can determine who has performed specific workflow tasks in a given workflow instance.
  - Input parameters: Workflow instance ID, task ID
  - (Pseudonymous) user IDs
- The general-purpose PDP answers policy requests that do not rely on history information.

- Input parameters: Applicable XACML policy, user attributes as a SAML assertion
- Result: Boolean value

7.4.3 A user performs a human task from their worklist

The next scenario again builds upon the scenarios discussed before: We assume that a human task instance has been deployed and that the user can see it in their worklist. Again, we list conceptual steps that are part of this scenario. Note that several steps are the same as in the previous scenario.

1. The user chooses one of the tasks in their worklist in order to perform it.
2. The worklist handler sends the task's ID and a reference to the user's identity information to the PEP-HT and *claims* the task for that user.
3. The PEP-HT retrieves information about the task instance from the PIP-Wf.
4. The PEP-HT sends a policy decision request with all of this information to the PDP-Wf.
5. The PDP-Wf reads the policy applicable to the workflow model in question from its policy store.
6. Using the policy, it determines other task instances relevant to the current decision request. Other task instances can be relevant because of binding-of-duty or separation-of-duty constraints.
7. The PDP-Wf determines who has executed these task instances by invoking the PIP-Wf.
8. The PIP-Wf sends a reply with the requested assignments.
9. It then evaluates the policy and answers the request. The sub-steps are the same as in the previous scenario.
10. If authorisation is granted, the PEP-HT registers the task as claimed. To this end, it sends a request to the PIP-Wf.
11. The PEP-HT also requests the attributes potentially accessed based on the task from the PDP-Wf.
12. The PDP-Wf sends a response based on the applicable attribute requirements policy.
13. The PEP-HT forwards the decision and the list of attributes to the worklist handler.
14. The worklist handler gets the user's attributes from the ZXID session and computes the intersection with the list of required attributes.
15. If the task has been claimed successfully, the worklist handler displays it to the user, accompanied by a notice listing the attributes potentially accessed by the workflow.

16. Now the user completes the task. The worklist handler notifies the PEP-HT.
17. The PEP-HT marks the task as completed in the PIP-Wf.
18. The PEP-HT then returns the result to the workflow.

The following new interfaces are used in the steps described above:

- The PEP-HT provides an interface for claiming a human task instance. This is similar to a simple authorisation request when viewing a task; in addition, this interface provides information about the potentially required attributes.
  - Input parameters: ZXID session ID, task-instance ID
  - Result: Boolean value (authorisation result), list of attribute names
- The PIP-Wf provides an interface for registering a human-task instance as claimed.
  - Input parameters: Task instance ID, user ID.
  - Result: None.
- The PEP-HT and the workflow have to provide an interface through which they can be notified of task completions.

  A notification of the completion of a task requires only one fixed parameter, namely the ID of the task instance. This is sent to the callback URL passed when creating the task instance. Most parameters, however, are determined by individual task definitions. We will dynamically create a WSDL interface description for each task definition.

  For the task definition given above, the body of the SOAP message would look as follows (again, we use the document/literal wrapped convention for SOAP calls):

```
<taskCompleted
   xmlns="urn:tas3:examples:taskdef1:completedNotification">
   <!-- ":completedNotification" added to the
        ID of the task definition -->
 <taskid>taskid-0815</taskid>
 <taskOutput>
  <x_or_y>y</x_or_y>
  <correct>true</correct>
  <flavour>forest</flavour>
 <taskOutput>
</createTask>
```

  As above, we look at all possible form elements in task definitions and consider how to transform them into schema elements:

  - For text inputs where `readonly` is not set to `true`:
    `<element name="`*id of the text input*`" type="string" />`

- For boolean inputs (checkboxes) where `readonly` is not set to `true`:
  `<element name="`*id of the boolean input*`" type="boolean" />`
- For a dynamic list of choices where `readonly` is not set to `true`:
  `<element name="`*id of the dynamic_choice input*`" type="string" />`
  (only the ID of the chosen option is returned).
- For a static list of choices where `readonly` is not set to `true`:
  `<element name="`*id of the static_choice input*`" type="string" />`
  (only the value of the chosen option is returned).
- URLs are always read-only.

Thus, for our example, the WSDL would look as follows:

```
<wsdl:definitions
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="urn:tas3:examples:taskdef1:taskCompletion"
   targetNamespace="urn:tas3:examples:taskdef1:taskCompletion" >
 <wsdl:types>
  <schema
    targetNamespace="urn:tas3:examples:taskdef1:taskCompletion"
    xmlns="http://www.w3.org/2001/XMLSchema">
   <complexType name="taskCompletionRequestType">
    <sequence>
     <element name="taskid" type="string"/>
     <element name="taskOutput">
      <complexType>
       <sequence>
        <element name="x_or_y" type="string" />
        <element name="correct" type="boolean" />
        <element name="flavour" type="string" />
       </sequence>
      </complexType>
     </element>
    </sequence>
   </complexType>
   <element name="taskCompletion"
            type="taskCompletionRequestType" />
   <element name="taskCompletionResponse">
    <complexType/>
   </element>
  </schema>
 </wsdl:types>

 <wsdl:message name="taskCompletionRequest">
   <wsdl:part name="root" element="tns:taskCompletion"/>
 </wsdl:message>

 <wsdl:message name="empty">
```

```
   <wsdl:part name="parameters"
              element="tns:taskCompletionResponse"/>
</wsdl:message>

<wsdl:portType name="taskCompletionPT">
 <wsdl:operation name="taskCompletion">
  <wsdl:input message="tns:taskCompletionRequest"/>
  <wsdl:output message="tns:empty"/>
 </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="taskCompletionSoapBinding"
              type="tns:taskCompletionPT">
 <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="taskCompletion">
  <wsdl:input><soap:body use="literal"/></wsdl:input>
  <wsdl:output><soap:body use="literal"/></wsdl:output>
 </wsdl:operation>
</wsdl:binding>

<!-- No endpoint defined because the operation has to be
     provided by the invoking workflow or proxy (PEP). -->
```

- The PIP-Wf also provides an interface for registering a human task instance as completed.
  - Input parameters: Task instance ID, user ID.
  - Result: None.

7.4.4 Effects of a separation-of-duty constraint on concurrent actions by several users

The following scenario is basically a modification of the previous one where not all steps run through "cleanly". Instead, it exemplifies what happens when authorisation is denied because constraints apply. The interfaces used here are the same as those that we have already discussed in detail.

1. Two human task instances t_1 and t_2 are created as described in Subsection 7.4.1.
2. A user views their worklist as described in Subsection 7.4.2.
3. The user chooses task t_1 in order to perform it.
4. The worklist handler sends the ID of task t_1 and a reference to the user's identity information to the PEP-HT and *claims* the task t_1 for that user.
5. The PEP-HT retrieves information about the task instance from the PIP-Wf.

6. The PEP-HT sends a policy decision request with all of this information to the PDP-Wf.
7. The PDP-Wf reads the policy applicable to the workflow model in question from its policy store.
8. Using the policy, it determines other task instances relevant to the current decision request. Other task instances can be relevant because of binding-of-duty or separation-of-duty constraints.
9. The PDP-Wf determines who has executed these task instances by invoking the PIP-Wf.
10. The PIP-Wf sends a reply with the requested assignments.
11. The PDP-Wf then evaluates the policy and answers the request. The sub-steps are the same as in the previous scenario.
12. The authorisation is granted, so the PEP-HT registers the task as claimed. To this end, it sends a request to the PIP-Wf.
13. The PEP-HT forwards the positive decision to the worklist handler.
14. The worklist handler displays it to the user.
15. The user interrupts their work on task t_1 and selects task t_2, which also appears in their worklist.
16. The worklist handler sends the ID of t_2 and a reference to the user's identity information to the PEP-HT and *claims* the task t_2 for the user.
17. The PEP-HT retrieves information about the task instance from the PIP-Wf.
18. The PEP-HT sends a policy decision request with all of this information to the PDP-Wf.
19. The PDP-Wf reads the policy applicable to the workflow model in question from its policy store.
20. Using the policy, it determines other task instances relevant to the current decision request. Other task instances can be relevant because of binding-of-duty or separation-of-duty constraints.
21. The PDP-Wf determines who has executed these task instances by invoking the PIP-Wf.
22. The PIP-Wf sends a reply with the requested assignments, namely that the user has already claimed the task t_1.
23. The PDP-Wf then evaluates the policy and answers the request. The sub-steps are the same as in the previous scenario. The result is that authorisation is *denied*, because the user has already claimed t_1.
24. The PEP-HT forwards this decision to the worklist handler, which displays an error message indicating that the user is not allowed to perform t_2.

A similar situation arises for two users when two tasks are connected by a binding-of-duty constraint. Here, the second user can no longer claim the second task once the first user has claimed the first task.

7.4.5 Invocation of a web service, including service selection

The next scenario is largely independent of the previous ones. However, it assumes that a human task instance related to a web-service call has been completed and the user who performed it is registered in the PIP-Wf. After this, the following conceptual steps occur:

1. The workflow triggers a web-service call and sends it to the PEP-WS, which acts as a proxy. This call includes the context of the call.
2. The PEP-WS sends the context to the PDP-Wf and requests the selection of a user whose identity is to be used for the call.
3. The PDP-Wf determines the corresponding workflow model ID for the workflow instance via the PIP-Wf.
4. The PDP-Wf looks up the applicable workflow policy.
5. The PDP-Wf retrieves relevant user task assignment history from the PIP-Wf.
6. Based on this history (which user has performed a human task related to the call?), it selects a user and returns the result to the PEP-WS.
7. The PEP-WS performs service discovery and finds all services of the respective types that are available to the user.
8. The PEP-WS performs service discovery to find the trust policy store of the user.
9. The PEP-WS retrieves the user's trust policy.
10. For each service, the PEP-WS retrieves a trust ranking from the Trust PDP.
11. The PEP-WS selects the service with the highest trust ranking.
12. The PEP-WS performs the actual call to the selected service.
13. The web service sends a reply.
14. The PEP-WS forwards the reply to the workflow.

Again, several interfaces are involved here. Here, we will only address those interfaces not yet introduced in the previous sections:

- The web-service call has to include the payload of the actual call, and context information:
  - The ID of the workflow instance.
  - The ID of the workflow activity making the call.

  - The type of service to be invoked.

- The PDP-Wf has to provide an interface for identity selection.
  - Input parameters: Workflow instance ID, workflow activity ID
  - Result: ZXID session ID

- The PIP-Wf has to make it possible to find out the workflow model for a given workflow instance.
  - Input parameter: Workflow instance ID
  - Result: Workflow model ID

- The discovery service determines which services of a given type are available to a user:
  - Input parameters:
    * Token identifying the user, directed to the discovery service
    * Service type, encoded as a URN
  - Result: Endpoint references to services of the given type, accompanied by tokens identifying the user and directed to the respective service.

- The trust policy store has to provide an interface to retrieve a user's trust policy:
  - Input parameter: Token identifying a user.
  - Result: Trust policy in a format understood by the Trust PDP.

- The Trust PDP has to provide an interface to determine the trust ranking of a service:
  - Input parameter: Endpoint URL of the service.
  - Result: Trust score (integer value, possibly negative) and whether the service is trusted according to the user's policy (boolean value).

7.4.6 Incoming web service call starts new workflow instance

Until now, all interactions were triggered by an event within the workflow management system. By contrast, we now look at a scenario that is triggered from the outside, by a web-service call sent to the workflow management system. It consists of the following steps:

1. An external entity sends a web-service message addressed to a web-service interface of a workflow. The respective interface is configured to start a new workflow instance (the corresponding WS-BPEL activity has the attribute `createInstance="yes"`).

2. The PEP-WS intercepts this message.

3. The PEP-WS extracts the identity information contained in the message.

4. The PEP-WS sends the payload of the message to the workflow engine, together with an identifier for the request.
5. The workflow engine determines the workflow model and the activity within it that the message is directed to.
6. The workflow engine creates a new workflow instance, which handles the web-service call.
7. The workflow instance sends a request to the PEP-WS, containing its ID, the ID of the activity that started the workflow instance, and the identifier for the original request created by the PEP-WS.
8. The PEP-WS determines whether the user is authorised to perform that activity, by invoking the PDP-Wf.
9. If not, it sends an error message in reply to the web-service message.
10. It also sends a reply to the workflow instance indicating whether the original web-service call was authorised or not.
11. Finally, the PEP-WS sends a request to the PIP-Wf in order to store the information that the user responsible for the original call has performed the starting activity of the new workflow instance.
12. Depending on the authorisation result, the workflow instance continues execution or immediately terminates.

Note that this scenario resembles the one already described in Subsection 6.4.9.

Two new interfaces are needed for this scenario. First, the workflow has to accept web-service calls that not only contains the payload originally sent by the external entity, but also additional context information. We address this issue later when we describe the workflow transformations necessary.

The other interface has to be provided by the PEP-WS. Through this interface, the PEP-WS gets information about the the workflow activities that processed the calls it forwarded to the workflow engine.

- Input parameters: Request identifier (generated by the PEP-WS and sent to the workflow engine together with the forwarded request), workflow definition ID, workflow instance ID, activity ID.
- Result: Authorisation decision (boolean value).

7.4.7 Deployment

In this final scenario, we consider the deployment of workflows, policies and human task definitions. This is actually a prerequisite for all the other scenarios.

1. A workflow developer creates human task definitions, possibly used in several different workflows.
2. The workflow developer then creates a workflow definition, incorporating human tasks and calls to external web services.

3. Finally, the developer writes the policies for the workflow, i.e. an access control policy, determining who may perform which activities in instances of the workflow, including separation-of-duty and binding-of-duty constraints, and an attribute requirements policy.

4. The workflow modelling tool deploys the security configuration to the PDP-Wf.

5. Then, it deploys the human task definitions to the worklist handler.

6. Finally, it deploys the workflow definition to the workflow engine. This is the last step to ensure that the corresponding access control policy and human task definitions are already available.

Quite obviously, three different deployment interfaces are needed:

- The workflow engine has to provide an interface for deploying workflow definitions.
  - Input parameters: Identifier of the workflow definition, workflow definition in WS-BPEL format.
  - Result: Acknowledgement of deployment.
- The PDP-Wf has to provide an interface for deploying access control policies:
  - Input parameters: Identifier of the workflow definition, access control policy, attribute requirements policy (both in the format described in Subsection 6.4.8).
  - Result: Acknowledgement of deployment.
- The worklist handler has to provide an interface to deploy human task definitions:
  - Input parameters: definition ID, human task definition as specified in Section B.1.
  - Result: Acknowledgement of deployment.

7.5 Optimisation of component interactions

Component interactions consisting of a large number of messages exchanged might indicate situations where performance can be improved, e.g. by changing the component where a particular kind of information is stored, or caching it in a component that frequently needs to access it. In the implementation of our architecture as described up to this point, a prominent example of such a performance bottleneck is the repeated authorisation of users to perform human tasks or activities in general (see Subsection 7.4.2, Subsection 7.4.3, and Subsection 7.4.4 above). The reason for performing authorisation repeatedly is that authorisation results may become invalid due to certain events.

A promising way to overcome the performance bottleneck while preserving decision responsibilities as they are is to introduce caching together with invalidation notifications. With this approach, authorisation decisions do not have

to be requested again as long as it is known that decisions received before are still valid. The following events may cause an authorisation result to become invalid:

- The attribute assertions regarding the user that were used for authorisation may expire. A possible solution would be assertions that are valid for a longer time, and notifications from the IdP when attributes are revoked. However, this would require additional infrastructure. Another solution would be caching the attribute assertions presented by the IdP the last time the user logged in and comparing them with the ones presented in the current session. Information from the PDP-Wf about which attributes were necessary for authorisation would make this solution even more viable. However, PDPs do not usually provide such detailed information on the reasons for an authorisation decision. In the end, the validity period of attributes is rather long (days at least) compared to user sessions (usually less than an hour). This mitigates this problem, so no further action is necessary.

- The login session of the user expires. This requires re-authenticating the user, but not necessarily repeated authorisation, provided that the attribute assertions provided by the user have not changed. Accordingly, the user's attributes from the previous login session have to be cached so that they can be compared with those in the new login session Only if the attributes have changed is authorisation repeated.

- The status of another activity related to the one for which authorisation has been performed by means of a BoD or SoD constraint has changed, i. e. it has been claimed or performed by some user. The solution here is to provide notifications of such events. The main question now is the granularity of such notifications. The first possibility is to provide notifications per workflow activity: Whenever the status of an activity changes, invalidation notifications are sent for all related activities and the respective human tasks. Because the number of human tasks currently active in a workflow instance is usually small (there are seldom more than two human tasks active simultaneously), a simplification is possible at little cost: Whenever the status of an activity changes, invalidation occurs for all activities of the same workflow instance. This solution can be implemented by extending only one interface: When the worklist handler sends a request to the PEP-HT that a human task has been claimed or completed, the PEP-HT includes a list of the IDs of human tasks for which authorisations are to be invalidated. For the PEP-HT to be able to do this, the PIP-Wf has to provide a new interface which makes it possible to look up all active human tasks of a given workflow instance.

7.6 Workflow transformations

To integrate workflows with the security infrastructure, the following workflow transformations are necessary:

- Outgoing web-service calls have to be redirected to the PEP-WS. In addition to the payload, the ID of the activity and of the workflow instance have to be included in the call.
- For incoming web-service calls, a request identifier is added to the payload. In the workflow, a loop with a request for authorisation has to be added (cf. Subsection 6.4.9).
- Requests creating human tasks have to be sent to the PEP-HT. The ID of the activity and of the workflow instance must also be added to this request. This transformation is similar to the transformation for outgoing web-service calls and not described separately.

7.6.1 Redirection of outgoing calls to the PEP-WS

SOAP messages are described in WSDL documents (Subsection 2.2.2, Subsection 2.2.3). They can consist of a number of parts. Non-payload information is usually encoded in headers. In detail, adding workflow context to SOAP messages in the form of a header requires the following transformations:

- An XML element for workflow context has to be defined. It consists of two sub-elements, `activity-id` and `workflow-instance-id`.
- The WSDLs of services invoked by the workflow have to be extended. To each request (i. e. outgoing) message, a *part* with the name `wf-context` is added, referring to the XML document introduced in the previous step.
- In the *binding* part of the WSDL, each `input` element of an `operation` element is modified. The binding declaration for the message body now has to contain the name of the respective part, because it is no longer evident which part becomes the body. A `soap:header` is added which refers to the `wf-context` part defined in the previous step.
- A BPEL variable holding the workflow instance ID is defined.
- For each `invoke` activity, a separate BPEL variable is defined. It has the same type as the `wf-context` part. This variable is initialised (using a `copy` activity) immediately before the `invoke` activity using the workflow instance ID from the respective variable and the activity ID as a literal value.
- If the `invoke` activity uses the `outputVariable` attribute, it is replaced by `fromPart` sub-elements of the `fromParts` child element of the `invoke` activity. A `fromPart` element is added in order to copy the contents of the respective variable to the `wf-context` part.
- `reply` activities are treated analogously.

7.6.2 Authorisation of incoming web-service calls

For the authorisation of incoming web-service calls, further modifications are necessary. The PEP-WS needs to add the request identifier in a SOAP header. In addition, a SOAP interface for transferring the authorisation result is needed.

This transformation has already been described conceptually in Subsection 6.4.9. The following description adds more technical details.

- An XML element must be defined for holding a request identifier. It contains only one value.
- The WSDLs of services *provided* by the workflow have to be extended. A part with the name `request-identifier` is added to each request (i. e. *incoming* message). This part contains a reference to the XML document introduced in the previous step.
- In the *binding* part of the WSDL, each `input` element of an `operation` element is modified. The binding declaration for the message body now has to contain the name of the respective part, because it is no longer evident which part becomes the body. A `soap:header` is added which refers to the `request-identifier` part defined in the previous step.
- For each `receive` activity, a separate BPEL variable is defined. It has the same type as the `request-identifier` part.
- Furthermore, for each `receive` activity, another separate BPEL variable is defined. It has the same type as the `wf-context` part.
- Finally, for each `receive` activity, another separate BPEL variable holding a boolean value (the authorisation result) is defined.
- If the `receive` activity uses the `inputVariable` attribute, it is replaced by `toPart` sub-elements of the `toParts` child element of the `receive` activity. A `toPart` element is added in order to copy the contents of the `request-identifier` part to the variable defined in the previous step.
- In addition, a temporary variable or several temporary variables are introduced for each `receive` activity. If the `receive` activity uses the `inputVariable` attribute, a variable with the same type as the variable referenced by this attribute is introduced. If it uses the `toParts` child element, a variable is introduced for each `toPart` sub-element, again with the same type as the variable originally used there.
- These temporary variable(s) replace the variables originally referenced.
- The `receive` activity (thus modified) is replaced by a structured activity as follows:
  - A `repeatUntil` activity is created.
  - Its child activity is a `sequence` activity.
  - The first child activity in the `sequence` is the (modified) `receive` activity.
  - Second, a `copy` activity initialises the variable holding the workflow context for the `receive` activity using the workflow-instance ID from the respective variable and the activity ID as a literal value.

- Third, an `invoke` activity calls an authorisation service provided by the PEP-WS. The variable holding the request identifier is copied to one part, the variable initialised in the previous step that holds the workflow context is copied to another part. The reply is copied to the respective variable.
- The condition of the `repeatUntil` activity is that the variable holding the authorisation result is *true*.
- Finally, a `sequence` activity is introduced. Its first child activity is the `repeatUntil` activity. After that, `copy` activities are created, which assign the received data stored in temporary variables to the variables where they would have been stored in the unmodified BPEL workflow.
- The `sequence` described in the previous step replaces the `receive` activity.

`pick` activities are treated similarly (although in a more complex way).

7.6.3 Creation of workflow instance IDs

Up to now, we have assumed that a workflow instance ID is created by the workflow engine as soon as the workflow instance starts, and that the BPEL code of the workflow instance can determine this ID. Unfortunately, this is only possible using extensions that are not part of the BPEL specification. For example, Apache Ode provides an XPath variable named `$ode:pid`.

An implementation-agnostic way of generating unique workflow instance IDs would be to let the PEP-WS do it. There are two ways of starting workflow instances, both of which we have to consider:

1. The first case is a workflow instance started by an external call to a special activity (one with `createInstance="true"`). This call is subject to authorisation, as described in Subsection 7.4.6. However, the interaction described there has to be modified slightly:

   - In step 7, the request does not contain the ID of the workflow instance.
   - Instead, if authorisation was successful, the PEP-WS creates a new ID and includes it in the reply described in step 10.
   - A new BPEL variable holding the workflow instance ID is introduced. In step 10, the ID is copied from the reply of the PEP-WS to this variable.

   In addition, the workflow transformation described in Subsection 7.6.2 is slightly changed:

   - There is no `repeatUntil` loop. Instead, if authorisation is denied, the new workflow instance is immediately terminated.

2. A user starts a new workflow instance through a dedicated console. In this case the applicable policy is known beforehand and enforced by the PEP-HT. Therefore, the new workflow instance is created immediately, without

the need to exchange several messages. The PEP-HT can directly create a new workflow instance ID and include it in the web-service message creating the new workflow instance.

7.7 Implementation of the PDP-Wf and integration of an XACML PDP

In Subsection 6.4.4.4, we decided to use an XACML-compliant stateless PDP, extended by a custom PDP for stateful decisions. In this chapter we have, up to this point, only mentioned one PDP-Wf. In Section 7.3, we already mentioned that a policy (separately defined for each workflow definition) has to consist of an XACML policy where the workflow activity is specified as a resource attribute, and additional history-based constraints. Subsection 7.4.7 only mentions deployment of the policy to the PDP-Wf.

Therefore, some details about about the separation of the PDP-Wf into two parts will be given in the following. We need to be able to distinguish between the XACML policies applicable to different workflow definitions during deployment. To this end, we can rely on hierarchically nested XACML `PolicySet`s. Workflow definitions are identified by an additional resource attribute. The global `PolicySet` contains one nested `PolicySet` for each workflow definition, which contains a `<Target>` element specifying the workflow definition it is applicable to. The part relevant to a given workflow definition is updated during the deployment of security configuration for that workflow definition. Only the history-based policy is deployed to the PDP-Wf proper. The PDP-Wf stores this policy (i. e. BoD and SoD constraints) in a relational database (indexed by workflow model ID and task IDs); the database assumes the role a a policy store.

When the PDP-Wf receives a decision request (through the interface introduced in Subsection 7.4.2), it first evaluates the history-based policy: It looks up the relevant constraints (i. e. those which belong to the workflow model ID and contain the task ID specified in the request). Through calls to the PIP-Wf, it determines whether the other tasks with task IDs occuring in the relevant constraints have already been performed or claimed in the current workflow instance (with the workflow instance ID also given in the request to the PDP-Wf), and if yes, by which user. The user ID of the user who needs to be authorised is determined via ZXID, using the ZXID session ID given in the request. If another task occuring in a BoD constraint has already been performed or claimed by a *different* user, or another task occuring in an SoD constraint has been performed or claimed by the *same* user, the request is denied immediately.

Otherwise, the stateless policy still has to be evaluated by the XACML PDP. The decision request to this PDP is sent using a ZXID library function which takes the user's attributes from the ZXID session with the specified ID; the workflow model ID and the task ID are added as resource attributes. The result is then forwarded to the caller (i. e. the PEP-HT or PEP-WS).

7.8 Summary

In this chapter, we have explained the implementation in detail. From a high-level component overview, we have derived detailed data formats and identifier schemes based on comprehensive interaction scenarios. This in-depth treatment shows the feasibility of the implementation. One important aspect is the workflow transformation, which enables compatibility with legacy workflows. We have broken it down to a level where it can be directly implemented. We have also shown that the performance of the implementation can be tuned without compromising the basic structure of the architecture, which is based on functional criteria.

Chapter 8

Evaluation

The goal of this thesis was to provide a flexible architecture with easily usable security mechanisms for a broad range of workflow-based applications. We will perform the evaluation with respect to this goal in two steps.

First, we evaluate how secure the architecture is. A naïve criterium would be the fulfilment of the functional security requirements from Chapter 5. This criterium, however, is not conclusive for several reasons. The requirements listed there are not from a single application. For this reason, they are rather high-level and abstract. In addition, it is already possible without the presented architecture to implement these requirements in some way or other. Therefore, the goal of designing the architecture was not to allow the implementation *in itself*, but to provide a more convenient way. Consequently, we deem the following criteria appropriate:

- With the architecture, is it easier to fulfil the security requirements than without it?
- Does the architecture help to avoid mistakes in the implementation of security specifications that might otherwise lead to security vulnerabilities?

Second, the architecture should be flexible, i. e. usable for a broad range of workflow-based applications. Obviously, not all requirements of all applications could be foreseen in the design of the architecture. Therefore, some extensions will likely become necessary to support new requirements. To evaluate how easily such requirements can be accommodated by the architecture, we consider several evolution scenarios. These scenarios encompass security requirements already part of the requirements found in Chapter 5, but not yet supported by the architecture, as well as advanced definitions of security properties from the literature. Finally, we present the practical results regarding one use case (user-centric audit) obtained in a user study.

8.1 Security of the architecture

We will now evaluate how secure the architecture is. Based on the technological baseline and the assessment of the state of the art in Subsection 5.1.4, we go through the requirements from Subsection 5.1.2 one by one. We analyse the

coverage of the respective functionality by our architecture and the extent to which the configuration mechanisms of the architecture help to avoid mistakes regarding security specifications for that functionality.

8.1.1 Technological baseline and state of the art

The technological baseline consists of the core web-services platform (Section 2.2) and BPEL for service orchestration (Subsection 2.3.2). In other words, it comprises the functional, i. e. non-security-specific parts of an architecture for distributed applications.

The state of the art considered in our analysis provides security mechanisms. We are interested only in such security mechanisms that can be used with the technological baseline (possibly with some adjustments). Recalling the results from Subsection 5.1.4.11, existing approaches address our requirements only partially. Moreover, most of them are not integrated into any WfMS.

8.1.2 Authentication requirements

In Subsection 5.1.2.1, we introduced requirements regarding the authentication of workflow models (R1), data (R2 and R3), service providers, requesters and users (R4–R10).

R1 is fulfilled by organisational measures and access control for the deployment of workflow models. R2 and R3 are supported by the integration of service discovery and trust evaluation (Section 6.3 and Subsection 6.4.6.2). These subsystems provide authentic information about services available to a user and about trust ratings of services. R4–R10 are fulfilled by the integration with ID-WSF. Fulfilling R9 introduces a privacy risk at first sight (which would violate R37), but this can be mitigated by certfication of workflow models.

8.1.3 Access control requirements

R2 and R3 (see previous subsection) also have an access control aspect. First, the integration of trust evaluation makes it possible to control based on trust policies whether data is transmitted to services and the results are used by the workflow. Second, the integration of user interactions for service selection ensures that the decisions of the user are respected regarding this. Below in Subsection 8.2.5, we describe an evolution scenario relating to the automatic performance of these user interactions. With this extension, workflow designers will be able to support service selection based on trust scores with great ease.

The integration with service discovery components provided by the architecture makes it unnecessary to perform all the necessary steps explicitly. In particular, a policy-based solution is used instead of hard-coded decisions. Thereby the security rule itself is separated from its enforcement, allowing it to be specified by specialists and to be reviewed indedently from the enforcement logic. This reduces the risks of implementation errors that would cause the disclosure of sensitive information to the wrong service providers or users, or decisions based on false information retrieved from untrusted sources.

The requirements in Subsection 5.1.2.2 (R11 to R22) concern access control for human tasks in workflows.

R11 and R19 essentially demand that other requirements (R15–R17 and R18 respectively) are also fulfilled when an activity that starts the workflow is involved. On the technical level, this may pose some problems when establishing the connection between a incoming message and a workflow instance. Subsection 7.6.2 describes how we can overcome these problems.

Automatic assignment of users to tasks (R17), taking into account policy constraints, and the evaluation of assignment constraints separately for each workflow instance (R14) are fullfilled according to the design of the architecture.

R12 is possible as part of binding-of-duty constraints (R18/R19). In addition, Subsection 8.2.6 shows how to support this requirement in combination with explicit assigmnent of users (R13) and controlled reassignment of duty (R20 and R21).

Finally, separation of duty R22 is supported as well, assuming that the identity management infrastructure always provides the same identifier for the same user. This holds for persistent SAML `NameID`s issued by the same identity provider if it is ensured that no user can have more than one account at that identity provider.

R23 to R28 concern access control for data external to the workflow management system but related to particular workflow executions. While our architecture does not directly accommodate these requirements, we explain in Subsection 8.2.3 how to do so based on extension points already foreseen.

Regarding web services, R29, R30 and R31 demand consideration of the trust requirements of users and the organisation (this part is already fulfilled by our architecture through the integration of a Trust PDP) and the involvement of users. The latter part is addressed by the evolution scenario in Subsection 8.2.5.

The automatic consideration of stateful access-control constraints avoids complicated logic for explicitly managing the required state information. Automatic assignment of available actors also increases the flexibility of the workflow execution. Finally, taking decisions based on policies is less error-prone than hard-coding the decisions in workflow logic.

8.1.4 Requirements regarding confidentiality, non-repudiation, integrity, and key management

Non-repudiation (R32 and R33) and confidentiality (R34 and R35) for incoming and outgoing web-service communication as well as key management (R44 and R45) are fulfilled through the integration with ID-WSF-based infrastructures and the corresponding configuration. The architecture thus enables implementations to rely on established high-level protocols instead of low-level cryptography, thereby automating the establishment of trust as well.

Confidentiality on the user interface level (R36) is a ensured via HTTPS encryption of the web interface. According to [ITU95b], integrity mechanisms have to protect against unauthorised modification, creation, deletion, insertion and replay of data. This has to be guaranteed on all levels of the system. Measures on the operating system and database level are not addressed by this thesis. Concerning the underlying protocols, ID-WSF requires per-message data integrity and transaction integrity mechanisms for the communication channel and messages [Lan03]. The integrity of data relevant to the workflow security mechanisms developed in this thesis (especially policies and runtime

context) is protected through encryption, digital signatures and access control mechanisms.

8.1.5 Audit requirements

R39 and R40 concern the recording of audit information, while R41 and R42 demand its accessibility in a user-centric way. While the former requirements are fulfilled by existing technology to some degree, the latter are not. Section 8.3 presents a prototype based on our architecture.

8.2 Evolution scenarios

In the following, we investigate several scenarios for the evolution of the architecture. The first two scenarios examine the integration of related work with our architecture. They concern concepts where the architecture presented in this thesis describes only basic solutions, but where more sophisticated solutions are available for specific aspects in isolation and described in the literature. We sketch approaches for integrating them with our architecture. The third scenario concerns access control for external data repositories that depends on the state of the workflow. These evolution scenarios demonstrate the flexibility of the architecture.

8.2.1 Delegation of Authority

One of our requirements is that the workflow must be able to invoke services on behalf of users (Subsection 5.2.2.5). This requirement, also known as delegation of authority, has been studied in depth in the literature. According to [Cha08], an important requirement on the conceptual level is the fine-grained decision of which privileges, attributes and roles are to be delegated. In addition, delegation should be subject to an express delegation policy. The author proposes a delegation-of-authority web service, which is invoked by users and issues delegation certificates. These certificates can be issued for an arbitrary period of time, can be used an unlimited number of times, and can be revoked at any time.

By contrast, we currently rely on the service discovery specified in ID-WSF. When a user logs into the worklist handler using SSO, the IdP supplies a token that makes it possible to use the discovery service and the identity mapping services. When a workflow needs to invoke a service with privileges of the user, it may have to use the discovery service in order to find a specific service endpoint. It then retrieves a token from the identity mapping service which allows the invocation of the desired service on behalf of the user. This token gives the delegate, i. e. the workflow management system, full access rights, exceeding the privileges required by the workflow, and thus the scope of the delegation as understood by the user. It is possible to ensure through manual audits that workflows only use these privileges according to the workflow's purpose and only after the user has expressed consent. However, this can not be enforced automatically and there are no explicit policies stating the privileges required by the workflow. Consequently, the privacy protection provided this mechanism is insufficient.

More controlled delegation and consequently better privacy protection can be achieved by integrating the above-mentioned delegation service into our architecture. To enable this, workflow designers would have to specify the following: Which kinds of services does the workflow need to invoke on behalf of users? Because specific service endpoints might only be known after service discovery, they cannot be part of the specification. Which data does the workflow access, and which actions does it perform on that data? For which period should the delegation be valid? However, this is difficult to determine in general. If the workflow takes longer, it might be blocked until the user renews the delegation. Furthermore, it can be difficult to exactly determine the privileges required by the workflow when it starts. On the other hand, users might be reluctant to delegate privileges that might not be needed. One possible solution would be to let workflow designers specify the required privileges separately for different branches of the workflow. However, this would require repeatedly asking the user for delegation, and the workflow would be blocked until the user actually performs it. It must be possible for permissions to be expressed generically (i. e. based on service *types* and *categories* of data) and to be delegated by the user based on such a generic definition. This requirement mainly affects the service provider, because they need to understand the generic permission definitions, but the delegation service must support the delegation of such permissions as well. There should be a web interface to the delegation service which is trusted by the user, and it should be possible for the workflow management system to send a request to this web interface (which the user can then grant or deny). This provides for a seamless user interaction.

8.2.2 Generic security architecture

We have described an architecture based on policy enforcement points for the interfaces of a WfMS and a dedicated PDP-Wf which can handle the state information to be used in access control policies for workflows. There exist generic proposals for security enforcement architectures. One such architecture is [CF12]. It separates the application-independent and application-dependent functionality of the PDP into different components, providing generic handling of, say, obligations and sticky policies. It also provides for a Master PDP aggregating the decisions of multiple PDPs. This current thesis by contrast deals with the special security requirements of workflows dealing with identity information, but does not propose a generic solution applicable to all kinds of applications. For example, the access control policy defined for the current workflow could be combined with sticky policies for the different user data accessed in the workflow. However, mapping the concepts developed here to a more generic architecture is a substantial effort and left as future work.

8.2.3 Access control for external repositories coupled to the workflow execution

Requirements R23 to R28 (defined in Section 5.1.2.2) basically demand that users involved in a workflow instance have access to (externally stored) data related to that instance. Their access rights have to be limited to particular data and must only be valid while they are performing certain tasks. For example, a clerk in a health insurance company processing reimbursement claims

for a customer's medical bills and checking them for plausibility should have access that customer's file, but only while actually processing the claim.

The technical difficulty here arises from the fact that the the WfMS and data repository are separate components. If data access by users is included in the workflow as a human task, the point in the workflow where access is possible is defined exactly, so there is no need to additionally define when access is *allowed*. However, there are good reasons to keep the workflow and data access separated: The data stored in the repository can have a complex structure, and there is not necessarily *one* precisely defined part of it relevant for the workflow. In knowledge-intensive workflows, such data often serves as background information for a number of tasks, and is not specifically assigned to a single task. Reducing the dependencies between the workflow and the data related to it also allows for their independent evolution. This aspect is particularly important in service-oriented architectures with multiple stakeholders.

A straightforward approach to implement this requirement is to use a delegation mechanism, as described above in Subsection 8.2.1: The user delegates access rights to external repositories to the workflow management system, which in turn delegates them to workflow participants charged with certain tasks. To this end, workflow designers have to specify the following: Which permissions have to be delegated? Again, it must be possible to express the required permissions generically. For which tasks are the delegated permissions necessary? This is used to determine both the delegate (i. e. the user performing these tasks) and the duration of the delegation (i. e. during some intervals encompassing these tasks). Note that these tasks should be subject to binding of duty. The WfMS has to ensure that delegations are granted to a user as soon as they are assigned to a group of tasks, and revoked as soon as that user (i. e. the delegate) has finished the respective tasks.

In order to ensure the timely revocation of delegation, the WfMS has to keep track of the execution of workflow instances. For this purpose we propose a new component called the *Workflow Delegation Manager* (WDM for short). The Context Handler, which we specified as a dedicated extension point in Subsection 6.3.4, can be used to provide the necessary information about the execution progress of workflow instances to the WDM. We have to additionally instrument workflow definitions so that the context handler is informed about the execution of tasks and can determine whether a group of tasks defined in the security specification of a workflow definition is currently active. The Context Handler then informs the WDM when a group of tasks becomes active or inactive. The WDM in turn uses the Delegation Service to cause delegations and revocations of the permissions delegated to the WfMS by the user. Its decisions are based on the specifications outlined above. The delegate is determined by querying the PIP-Wf is assigned to one of the tasks in a group of tasks referred to in such a specification.

8.2.4 User-centric audit of workflows

Requirements R39 to R43 (defined in Subsection 5.1.2.6) demand the generation of meaningful audit events and their presentation to affected end users (data subjects).

The workflow reference model includes an interface for audit information. The WfMC has specified a format for audit messages [WfMC98]. This format

is focused on states of workflow instances and activity instances and does not address the handling of data items and the relationship between different workflows through message exchanges. This makes it insufficient for our purposes. We see two alternatives for providing audit information: (1) We could rely on the audit events generated by the BPEL engine. However, [WfMC98] is insufficient for our purposes. But without a widely supported standard format, this would require separate implementations for every BPEL engine to be supported (either by adding event generation to it, or by handling the format used by that engine). In addition, formats used by existing BPEL engines do not include the necessary information either. (2) The workflow definitions written can be instrumented. This means that workflows are modified so that they generate audit events. A similar approach is pursued in [YCW+10]. It needs to be generalised in order to include all information necessary. – Because of the wider applicability and the likely easier implementation, we advocate (2).

The integration is based on one component of the architecture, the context handler. It has to provide different kinds of audit information. Some of this information is already available to components of the WfMS, while some has to be specifically captured for audit purposes:

- Activities performed and branches taken can be provided to the Context Handler by instrumenting the workflow definition: For each activity, a call to the context handler is added.

- The user who performed a given task is already known to the PEP-HT. Therefore, once a task has been completed, this component should inform the Context Handler which user performed it.

- The invoked service is known to the PEP-WS. Like the PEP-HT, it should inform the Context Handler of the service invoked.

- Determining the data processed in each workflow activity is more complex. Assuming that the payload of web-service calls is accompanied by respective meta data, the PEP-WS can extract it and provide it to the Context Handler. For human tasks, workflow definitions can be annotated with a description about the data used in each task. Then the Context Handler can determine which data was used in a completed human task.

All this information has to be provided to the audit application immediately so that the audit can be performed in real time.

8.2.5 User interaction for service selection

R29 (Section 5.1.2.2) requires the possibility for users to confirm the selection of services invoked as part of a workflow. Doing this manually is tedious and error-prone. Therefore, the authors of [MvSB11b] and [MvSB11a] have developed the concept of *user involvements*, i. e. workflow fragments for security-specific user interactions that are automatically inserted as a result of certain annotations.

For service selection, the following steps have to be performed:

- The list of available services for a given service invocation is retrieved.

- The list is presented to the user, who chooses one of the services or declines the invocation.
- If the user chooses a valid entry, the respective service is called.

A straightforward way to implement these steps with our architecture is as follows:

- The PEP-WS provides a (new) interface to perform service discovery (including determining trust scores) for a given service type, and to retrieve the list of service endpoints and their associated trust scores.
- Further, the PEP-WS provides another new interface to set the selected service for a certain service (identified by activity name and workflow instance ID).
- A human task definition is added to workflow definitions, allowing the user to select a service endpoint from a list.
- Workflow definitions are instrumented as follows: Before a web-service call activity (with the respective annotation), several workflow activities are added: First, the list of services is retrieved from the PEP-WS. Then, a human task is created to allow the user to select a service, and the workflow waits for completion of this task. Finally, the service selection is set in the PEP-WS.

Note that a binding-of-duty constraint should be specified between the web-service call activity and a human task, so that the system can determine which user will perform the service selection.

8.2.6 Explicit user-task assignment and reassignment of duty

We have formulated two requirements calling for explicit assignment of users to tasks. R13 is the more general one. It demands that it must be possible to determine the user responsible for a human task by the explicit decision of another user, or by some workflow logic. R20 treats a specific case: For several tasks connected by a binding-of-duty constraint (for which such a transfer of responsibility has been allowed), it should be possible to transfer the responsibilility for the remaining tasks from one user to another. In both cases, the assigned user has to fulfil the applicable policy. For the latter requirement, R21 explicitly demands this.

In order to implement these requirements, we propose introducing unique identifiers for groups of task connected by BoD. Note that binding-of-duty is an equivalence relationship and thus induces a partitioning of the sets of all activities of a workflow definition. Tasks for which no BoD constraint has been defined form their own partition (with cardinality 1).

Second, we need the possibility to determine available users. We assume there is a directory to this end, making it possible to check the applicable policy separately for each user. Note that the attributes of these users must be known to the PDP-Wf as well (or it must be able to retrieve them). Thus, the

PEP-HT can provide an interface to determine eligible users for a given group of tasks, and to assign one of these users.

In order to reassign the responsibility for groups of tasks, it is convenient to know which of them are still active. The data structures in the PIP-Wf must be changed so that it is possible to assign a user to a entire *group* of tasks. It must be possible to determine who is currently assigned to such a group, who has performed the individual tasks, and whether the group of tasks is currently active. We consider a group of tasks active when there are still tasks to be performed. This can be determined using control flow analysis. At the points in a workflow definition where a group of tasks becomes inactive, calls to the PEP-HT have to be inserted. The PEP-HT can thus keep track of the status of a group of tasks and update the data in the PIP-Wf accordingly. The status can also be used to enable and disable delegated privileges connected to a group of tasks (see Subsection 8.2.3 above).

To perform the actual reassignment, a user interface is necessary. In Subsection 6.3.4, we introduced the *Async UI* component, which is suitable for this purpose. When a user accesses this interface, it can determine the groups of tasks currently assigned to them. When a user chooses to reassign responsibility for one of these groups, it can retrieve a list of eligible users from the PEP-HT and perform the reassignment if necessary.

8.3 User Study: User-Centric Audit based on the Architecture

Until now, we have focused on the technical aspect of secure workflow management. We now show that it has an application with a real benefit for users. To this end, we present a real-world application of workflow security[1] and an evaluation with users.

8.3.1 Motivation of the scenario

Earlier in this chapter (Subsection 8.2.4), we showed that the workflow management system can provide audit information. This enables applications allowing the user to understand the workflow execution and its security and privacy aspects. We explain the relevance of such applications in the following.

Today, companies outsource parts of their processes to other companies that can perform them more efficiently. As an example, think of a loan approval process: Consumer loans are highly standardised products with low profit margins. Banks and other organizations granting loans share information about risk factors and credit defaults through specialised credit bureaus like SCHUFA in Germany. As the first step of the loan approval, the creditor queries risk information from such an agency, then calculates the interest rate. This calculation can be outsourced as well. If a credit default occurs, information is sent to the credit agency. This example shows that outsourcing leads to personally identifiable information (PII) being transferred to third parties. PII is protected by law in the European Union [EC95] and elsewhere. The law gives individuals (data subjects, i.e. the person the data relates to) the right

[1] The contents of this chapter have been published in [MKB12a] and [MKB12b], based on the B. Sc. thesis of Murat Kavak supervised by this author.

to request information on how their data is processed, and where it is transferred. It also requires the informed consent of individuals to any processing and transfer of PII that is not necessary for the service provided. In order to give this *informed* consent, the user must be able to assess how his or her data will be used.

Current mechanisms fulfil the law formally, but are not useful in reality. Companies usually require individuals to give consent by signing terms and conditions that are both very broad and detailed. Information on data processing that companies provide is normally in text form. With large amounts of text, it is hard for users to find the details they are interested in. When PII is processed in a distributed way, companies have to tell individuals which other companies they have transferred it to, or where they have acquired it. In principle, this allows users to ask the other companies for information on their data. In practice, this is too tedious and time-consuming for users, especially when they have to follow their data over multiple hops. We conclude that information must be structured in a way that is easily accessible to individuals.

Workflow management supports the complete lifecycle of orchestrations, i. e. applications combining lower-level functionality, from models to executable workflows. Non-functional requirements, including security requirements, can be expressed as annotations to graphical workflows models [MvSB11b].

The goal of this section is to study how to let users track workflows that use and transfer their data with ease. Because the right to information is not limited to finished cases, users must be able to get information about running workflows as well. This process in general is known as *auditing*. A workflow management system can automatically generate the necessary events at runtime. To design auditing tools, we also need to understand how real users work with them. All these tasks are challenging, for various reasons at different levels:

- We need a succinct representation of audit information that is easy to understand at first glance. A system of this kind should also allow for drill-down in order to get more details. In particular, it should be easy to switch to another workflow following the data flow. Such a representation is not readily apparent.
- It is not possible to determine which auditing features are useful for users without a realistic scenario they themselves are part of.
- The functionality envisioned should reuse artifacts created anyway when modeling and deploying a workflow. This minimises the additional effort for application developers. This point concerns two issues in particular: First, audit information must be presented to the user. This step can reuse graphical workflow diagrams. However, additional information about their structure is needed. Second, the audit tool must acquire the necessary information at runtime. This requires an interface between the audit tool and the security components of the WfMS. However, the implementation is not readily apparent, as we will explain.

To this end, we have designed and implemented an auditing tool dubbed WoSec (*Wo*rkflow *Sec*urity) and have evaluated it with real users. More specifically, our contributions are as follows:

- We have analysed which information must be provided to users in order to audit data transfer in distributed applications, and how it can be presented visually.
- We have developed WoSec, a web-based tool for auditing the handling of PII. It works with graphical representations of BPMN models of applications. It allows users to "follow their data" when it is transferred to another organisation that also provides data to WoSec. It can also be used to visualise how an organisation *intends* to handle PII, enabling users to give more informed consent.
- We have designed several sample use cases for distributed data processing that are sufficiently complex for a realistic evaluation: applying for an internship, trading items on an online marketplace, and buying a car.
- We have designed a user study for evaluating our tool and various features of it. Having carried out the study, it shows that users prefer graphical audit facilities, and that these lead to a better understanding of data transfers. We have discovered that usability is curbed severely when workflow diagrams do not fit the screen without scrolling. This finding as well as other ones lead the way to an improved version of our audit facility.

8.3.2 Related Work

Weske [Wes07] mentions a monitoring component that visualises the status of workflow instances, but makes no statement on how such a visualisation should look. The Intalio Designer *BPMS Console* lists available workflow definitions and instances and marks activities currently running. However, it does not address data transfer and is targeted at administrators, not end users. To the best of our knowledge, the effectiveness of generic audit facilities for data processing dedicated to end users has not been studied or empirically evaluated in the literature.

8.3.3 Functionality

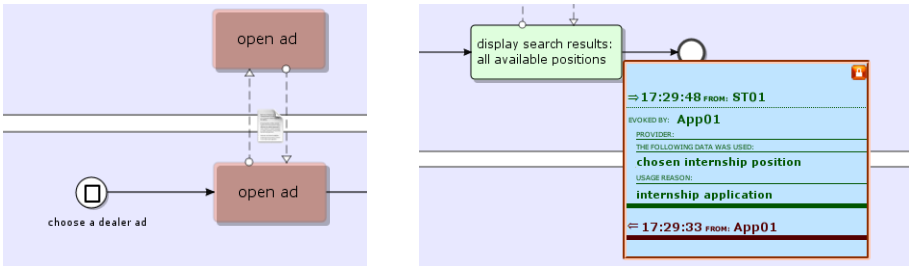
In the following, we present requirements with respect to the functionality of our audit tool from the user perspective. We have derived them by systematically inspecting which kinds of data that users are interested in arise in real-world workflows. This description serves as a basis for our proposal of an architecture able to provide this functionality.

We can explore this in more detail by taking a user-centric approach and considering a set of workflows handling one person's data This individual is allowed to track how their data is processed. To this end, they can access a list of workflow instances handling their data, and a detailed audit view for each. This view contains historical and live information, as we will explain below.

General structure: Because BPMN is a generally accepted standard, and BPMN diagrams already exist for applications modelled as workflows, we have decided to use them as the basis for visualising audit information. Note that such diagrams are static and do not contain information on the current state of workflow instances. They contain lanes (horizontal) representing the

workflow (coordinating the overall application behaviour) and roles involved in it as well as external parties, activities (rectangles), solid arrows (mostly horizontal) representing the control flow, and arrows (mostly vertical, dashed) representing the data flow between the workflow and persons/external parties.

Execution progress: WoSec visualises execution progress by highlighting activities already executed and currently executing in this diagram, using a different color. When an activity starts execution, the tool automatically scrolls the viewport to that activity and notifies the user acoustically. Detailed information about activities is available, such as the time when they were performed and the user who performed them.



(a) Animation of data flow (b) Information box with details about an activity

Figure 8.1: Screenshots of WoSec

Data transfer: A data object moving between activities indicates data transfer (Fig. 8.1a). An information box provides details about the data transferred (Fig. 8.1b). This box is shown when pointing at activities and contains a colored mark indicating whether the activity sends or retrieves data, the external party involved, and the data itself, possibly with attachments displayed as links. The audit view allows access to historical information as follows: A timeslider lets the user move to some point in the past and and replay events from there. In the case of loops, activities can be executed more than once. Information boxes for activities also contain information about earlier executions of an activity, not just the most recent one.

Multiple diagrams: When multiple organisations are involved in a workflow, they model their respective part independently, so there is no overall diagram for the whole workflow. Instead, several diagrams show the perspective of each participating organisation, where the workflows of other organisations are only represented by their interfaces called by the current workflow. This leads to smaller diagrams that are easier to take in. It must be easy for users to discover what happens with their data in the other workflow. WoSec accomplishes this as follows: When an activity transfers data to another workflow, the user can jump to the other workflow instance from there. WoSec then opens it in another view and scrolls to the activity which receives the data.

Color scheme: As we add information to the basic BPMN diagrams by changing the colour of elements, we need a colour scheme in line with our intentions. We start out with diagrams that use the colour scheme of Intalio BPMN Designer[2], i. e. white tasks on a very light blue pool background. Our

[2] `http://www.intalio.com/bpms/designer`

Event type	Meaning
EventCommand	to initialise a workflow instance
StartingTask	to mark a task as active
FinishingTask	to mark a task as finished
SpecifyingParticipant	to specify the communication partner
TransferringData	for data transfers between two active tasks

Table 8.1: List of event types the server sends to the client

goal is that active tasks stand out, finished tasks retreat into the background, and information boxes are prominently visible, even compared to active tasks. This leads to active tasks in red, finished tasks in light green, and information boxes in a saturated light blue.

8.3.4 Architecture and Design

We now describe the architecture of our application that yields the functionality presented in Section 8.3.3. It has to address three main issues: First, the WfMS has to provide information on the execution of workflow instances to the auditing tool. We have addressed this issue in Subsection 8.2.4. Second, workflow diagrams that have been created in order to be translated into executable workflows must be provided to WoSec in a suitable form, i. e. a mapping from activities to graphical elements must be created. Third, we need to develop the internal architecture of WoSec.

Creation of graphical workflow models: Domain experts and workflow modelers initially create workflows as BPMN diagrams in a graphical modeling tool. These models are translated into executable BPEL workflows. It is possible to instrument them automatically to provide audit information. Then the models are deployed in a BPEL engine. From the same BPMN diagrams, we need to create graphical representations for our auditing tool. It must be easy to highlight elements in them, and it must be clear which elements belong to which workflow activities. Intalio BPMN Designer produces an SVG version of workflow diagrams where elements are annotated with activity names. We have to create a description file for the mapping between activities and graphical elements. The SVG graphic and the description file are then deployed to WoSec. Figure 8.2 shows how the auditing tool fits in the overall WfMS architecture.

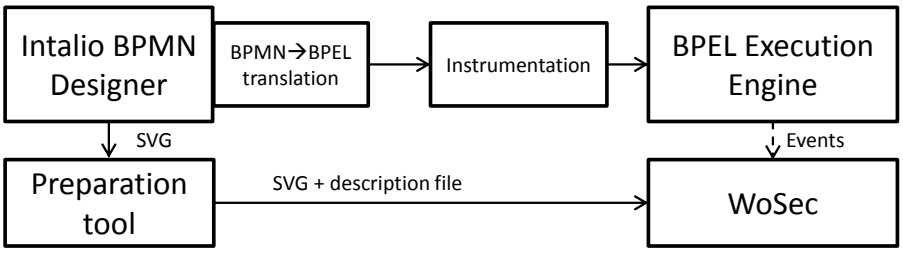


Figure 8.2: Integration of the auditing tool into the overall WfMS architecture

Internal Architecture of WoSec: WoSec itself displays audit information by overlaying it on the SVG diagrams. We have taken the following design decisions: (1) We have chosen a client/server design because we need a server part that stores workflow definitions and is able to receive audit events at any time. In general, an application running on the user's computer cannot achieve this. (2) The client can either be a stand-alone application or a web application running in a browser. This also brings up the question of where to handle the graphical representation. Because a web application does not require installing any software and is thus easier to use, and modern browsers natively support SVG, we have pursued this option. (3) A question in any client/server application is the granularity of information transferred between client and server. In our solution, all event information is cached on the client side and thus sent only once per login session to minimise response times. To be able to display the events, the following information is provided for each event: an event type (see Table 8.1), a timestamp, the id of the SVG element the element applies to, a list of all entities involved, attachments sent, and a list of data used and the purpose of use.

8.3.5 Study Design

The goal of our study was to find out whether our visualisation improves users' understanding of how their PII is processed. In addition, we want to discover shortcomings of WoSec, possible misconceptions of how audit information should be presented to users, and consequently opportunities for improving WoSec in particular and audit facilities for end users in general.

To achieve these goals, we need an appropriate scenario and a realistic baseline for comparison. We also need to measure measure participants' understanding. This requires asking participants for their assessment according to criteria we deem important. In addition, we need to check these answers for plausibility. We do this by asking questions about what has happened in the scenarios and analysing participants' behaviour during the study. We use textual audit messages as the baseline, as organisations still use these to answer information requests and fulfil their legal obligations.

The scenarios used in the study must be sufficiently complex. This means that several organisations should be involved in processing different kinds of data. The scenarios must match the interests of the study group. Although they are less detailed than in the real world, they must appear as natural as possible.

We provide a realistic web application and the corresponding audit view at the same time, using a split browser window. The upper half shows the mock-up of a web application, while the lower half contains an audit view, which either uses our visualisation or is text-based.

We chose to evaluate an early prototype of WoSec in order to get preliminary feedback. Subsequently, we improved it and then carried out a more extensive and sophisticated study. In both studies, we compared our graphical auditing tool to a text-based audit view. Before working through the scenario, we asked the participants questions about their Internet usage and some demographic information. In each round, the participants worked with one scenario and one variant of the audit view. After each round, we asked the participants assessment questions, as well as control questions about the scenario. Finally,

we asked them to compare the different rounds. The full questionnaires can be found on a dedicated website[3].

8.3.6 Pre-Study

The early prototype was tested with individuals who are IT users, but who do not have a significant computer-science education. The participants were 16 high-school students. The scenario was a fictitious social network called "FaceVZ". Such a scenario is well-known to the target group and, according to our expectations, triggers increased privacy awareness due to recent media reports about privacy threats in such networks. The scenario included several steps: Images are uploaded to a social network, face recognition linking the pictures to accounts is performed, and finally free prints of the pictures can be ordered. We used two versions of the scenario, with a subtle difference: In one version the user address was submitted to the print service, in the other the prints had to be picked up in a shop. All participants used both the graphical and the text-based audit. To rule out learning effects, the participants were split into two groups, each of which used the two versions in a different order.

This pre-study revealed some shortcomings in the implementation, leading to visualisation errors. One key result was that the participants liked the graphical version better. When participants were asked to decide between versions on a 7-step Likert scale where 1 indicates a strong preference for the textual version and 7 for the graphical version, they preferred the graphical version with respect to user-friendliness (mean: 5.69) and clarity (mean: 5.13). The group that tried out the graphical version first was able to give about 50 % more answers to the control questions. Unfortunately, the participants did not fully understand the purpose of such an audit system, as they were not able to give any examples of possible applications of such audit systems.

8.3.7 Main Study

For the main study, we used an improved version of the visualisation with more features. We tested the following hypotheses:

- H1: Our visualisation helps users to understand which of their data is transmitted to which organisations.
- H2: The visual audit facility has good usability.
- H3: Users prefer a graphical audit facility over a text-based one.

Because the participants of our pre-study were not able to really see the benefits of an audit facility, presumably because of their young age, we chose to perform the main study with more experienced participants. To evaluate the functionality of WoSec in full, we devised a more sophisticated study setup. We decided to use two different configurations of the visualisation, in addition to the text-based audit view: While both configurations allow access to all audit information available, one contains additional features aimed at improving usability. This makes it possible to test the influence of the non-essential features

[3] `http://dbis.ipd.uni-karlsruhe.de/english/1746.php`

and explore the opinion of the participants regarding the additional functions. To avoid learning and order effects when comparing the configurations, we used different scenarios for the different configurations. The participants went through the scenarios in the same order, but the order of configurations was randomised.

In both configurations, the visualisation was immediately updated when something relevant had happened in the web application. Both contained the basic animations, i. e. activities starting execution blink, and animation of data transfer. In addition, the full-featured configuration automatically scrolls to activities that are becoming active. It also shows the actual data transferred, the user causing the activity, and the reason for transferring the data. For activities, a window shown on right-click contains a textual description of the activity and all users involved.

We recruited our study participants from our directory of individuals interested in user studies related to information systems, which includes mostly university students and adults with university education. We designed three scenarios specifically for this group:

(1) *Internship application:* In this scenario, eligible students are supported in applying for placement in an internship program relating to their course of study. First, participants enter their registration information. They then wait for several steps: A university coordinator has to approve their application, registration data is written to a database, and the university coordinator chooses a placement service, which then sends a list of possible internships. Finally, the participant chooses an internship. (2) *Online trade:* In this scenario, participants have to sell an item. First, they have to enter their trader data and the item description. They then wait for another customer to open the ad, buy the item and send their contact data. The participant now receives the address of the buyer, has to prepare the parcel and hand it to a parcel service. (3) *Car purchase:* Here, the participants have to buy a car on credit. They have to wait for a list of available cars and choose one. They then have to enter their personal data and, in our example, state that they want to buy the car on credit. They choose a bank for the loan and accept the terms of the credit bureau. They then have to wait for several steps happening in the background: The bank receives the credit application and receives a score from the credit bureau. We assume that it grants the application. The car dealer receives a confirmation from the bank, reserves the car chosen and sends a purchase confirmation to the customer.

We paid the participants €10 for their participation. To incentivise active participation, we promised an additional amount based on the level of participation. We computed this amount based on the number of questions answered. This means that participants who answered all 60 questions were paid another €5. In addition to the questions answered, we recorded for which scenarios, tasks, and participants information boxes for activities were shown. To get an overall impression how the user interface was used, we recorded a so-called heat map, overlaying mouse clicks onto a screenshot of the user interface.

8.3.8 Results

In total, 17 individuals participated in our study. The study group included participants of different ages (20–74 years). All of them had some technical

background, and all expressed some privacy concerns regarding Internet usage.

We performed the study in two separate meetings with participants, with 7 of them in the first and 10 in the second one. Due to technical problems, we were only able to test the full visualisation at the first meeting. This means that 7 participants used the full visualisation for all three scenarios. We weighted them with $1/3$ when computing the mean values to achieve the same weight per participant.

We compared the ratings the participants gave the text-based and graphical audit view, as well as the restricted and full graphical version. We first tested all samples for normal distribution with the Shapiro/Wilk test [SW65]. Because this test did not confirm a normal distribution, we had to use the Wilcoxon/Mann/Whitney test to compare samples. The average answers and the result of the significance tests are shown in Table 8.2. For all tests, we required a level of significance of $\alpha = 0.05$. We performed one-sided tests of whether the underlying random variable of the sample with the larger mean was actually significantly larger. Table 8.3 contains the mean values of the questions asking for a direct comparison. The samples did not show a significant difference from the neutral value 4.

In Section 8.3.5, we hypothesised that our visualisation helps users to understand which of their data is transmitted to which organizations. (H1) The statistically significant difference for Q1, Q2, Q3, Q8 and Q9 shows that the visualisation indeed led to a better understanding of data transfers. Moreover, the participants were slightly more satisfied with the amount of information available in the visualisation (Q11), although the difference was not significant.

Regarding H2: *The visual audit has good usability*, we cannot show any statistical significance for Q5, Q6, Q7 and Q12. This indicates that the usability of all audit systems is equal. Nevertheless, the restricted visual audit scores significantly better than the full one. We conclude that automatic changes of the viewport decrease usability.

As one may expect, we could not show any statistically significant difference for H3, *Users prefer a graphical audit facility over a text-based one* through questions Q10, Q11, Q12 and Q13. Yet, except for a small outlier (Q10), the participants evaluated the visual audit slightly better than the textual audit.

In total, our tool improves users' understanding, but there is potential for better usability. In particular, participants were annoyed by automatic scrolling. On the other hand, some participants noted that the size available for the diagram was too small, and the heat maps showed that the scrollbars were used a lot. We conclude that automated scrolling should be improved, or scrolling should be made unnecessary. We believe that splitting diagrams into parts and easy navigation between the parts can alleviate these problems. Participants mostly accessed the additional information provided by the information boxes when answering the questions. This indicates that they had used the audit facility mainly for that purpose. In a future study, participants should answer the control questions solely based on their usage of the system up to that point.

8.3.9 Conclusions

Using our architecture for a secure workflow-management system, we created a graphical tool that allows end users to audit workflows involving data trans-

	G	T	F	R	G/T^s	R/F^s
Q1 Have you been able to trace the flow of the data?^{S1}	5.71	4	5.31	6.1	$>$	$=$
Q2 Have you been able to trace why a data flow has happened?^{S1}	5.99	4.3	5.78	6.2	$>$	$=$
Q3 Could you predict following steps?^{S1}	5.43	4	5.26	5.6	$>$	$=$
Q4 How was the number of animations?^{S2}	4.8	3	4.6	5	$>$	$=$
Q5 Have you been able to orient yourself without problems?^{S1}	5.01	5.2	4.51	5.5	$=$	$=$
Q6 How clear was the auditing tool?^{S3}	4.61	4.1	3.92	5.3	$=$	$>$
Q7 How user-friendly was the auditing tool?^{S4}	4.6	4.3	4.39	4.8	$=$	$=$
Q8 How much information content did the auditing tool contain?^{S5}	5.34	3.9	4.63	6.13	$>$	$>$
Q9 Do you feel adequately informed about all actions?^{S1}	4.93	3.9	4.75	5.11	$>$	$=$

Legend:
G : all graphical versions T : textual version F : full graphical version
R : restricted graphical version s : significance test
S1: 1 = absolutely not 7 = absolutely yes
S2: 1 = too few 7 = too much
S3: 1 = not clear at all 7 = absolutely clear
S4: 1 = not user-friendly at all 7 = absolutely user-friendly
S5: 1 = very little information content 7 = very much

Table 8.2: Assessment of different audit configurations

	Mean
Q10 How clear was the textual visualisation compared to the graphical visualisation?^{S1}	4.8
Q11 How much information content did the textual representation contain compared to the graphical visualisation?^{S2}	3.5
Q12 How user-friendly was the textual representation compared to the graphical visualisation?^{S3}	3.3
Q13 Which visualisation do you prefer?^{S4}	4.5

Legend:

S1: 1 = very unclear	7 = very clear
S2: 1 = very little	7 = very much
S3: 1 = not user-friendly at all	7 = very user-friendly
S4: 1 = strongly prefer textual	7 = strongly prefer graphical

Table 8.3: Direct comparison of text-based and graphical audit

fer, and described its integration into a WfMS. We have carried out a study comparing the tool with text-based audit facilities which represent the current state of the art, and assessed its impact on effectiveness and usability. Next to other points, the results show a usability problem related to limited viewport sizes and automatic scrolling. However, the design of the graphical auditing tool and the implementation of a prototype demonstrated that our secure workflow-management system is well-suited for such an application.

8.4 Summary

As shown in this chapter, the architecture as originally described already fulfils many security requirements of workflow management systems, using built-in functionality and configuration mechanisms that help in avoiding incorrect security specifications. It is also flexible enough that it can be extended to meet the remaining requirements. Moreover, it enables applications with practical benefits, as demonstrated.

Chapter 9

Summary and Outlook

9.1 Summary

At the beginning of this thesis, we provided a comprehensive presentation of the fundamentals of the field (information security, workflow management, and service-oriented architectures) and related work. We then introduced use cases based on real applications that were developed as part of the TAS^3 research project. Based on this, we performed a detailed requirements analysis, well-structured according to well-known aspects of security and workflow management. A comparison with the state of the art showed that these requirements were not sufficiently supported before, especially not in combination and in the dynamic setting we face today. The requirements analysis was continued in greater detail on a more technical level concerning integration with a framework for federated identity management.

We then designed a system architecture capable of fulfilling the requirements. This was followed by a detailed description of important implementation aspects: the division into individual components, data formats and identifier schemes, the interfaces used for interaction between the components, and the approaches used for integrating components that implement existing standards.

Finally, we evaluated the security requirements, i. e. whether the architecture and the configuration mechanisms it provides make it easier to fullfil these requirements and help avoid mistakes. We also analysed the architecture's capability to accommodate additional requirements by means of evolution scenarios. The results show that the structure of the architecture is well-suited for the requirements originally described as well as additional requirements. Finally, a user study involving a practical scenario supported by the architecture demonstrated its real-life benefit.

9.2 Outlook

Further research perspectives arise from the architecture and its implementation, and most importantly from the evolution scenarios presented in Section 8.2. We can divide them into three main categories:

- On the *technical* level, standardised protocols and policy formats, performance and deeper technical integration are important topics. For ex-

ample, a standardised way of referring to workflow activities in policies would promote interoperability between tools for workflow definitions and policy decision points. Interoperability is also important for higher-level policies, especially those responsible for coupling access control for external repositories with the workflow execution.

- On the *conceptual* level, the relationship of access control for workflows and delegation should be investigated more closely. This concerns especially access control for external repositories and the explicit assignment of users to tasks or group of tasks. In both cases, a delegation policy can support fine-grained control regarding which transfers of permissions are permitted. Further investigation is also warranted regarding workflow-specific concepts for more generic security architectures. This concerns in particular the handling of stateful access-control policies and the usage of obligations to perform generic tasks such as recording audit information or deactivating access permissions.

- Regarding *user interaction and user interfaces*, the most efficient and most user-friendly route for security-specific user interactions should be studied. This concerns functionality such as service selection (already treated in Subsection 8.2.5) or the revocation of permissions. Especially when users should be able to interact with the workflow management system at any time during workflow execution (we have referred to such user interactions as *asynchronous*), further exploration is necessary as to how the user can access such functionality easily. The classical user interface of workflow management systems based on a task list should also be studied further in order to provide a more flexible user experience.

Appendix A

BPMN Model of the APL Scenario

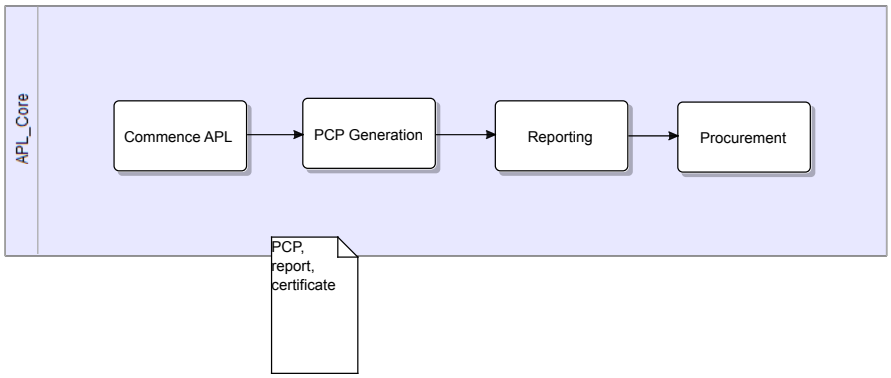


Figure A.1: BPMN Model of the APL Scenario: Top-Level Workflow (Core)

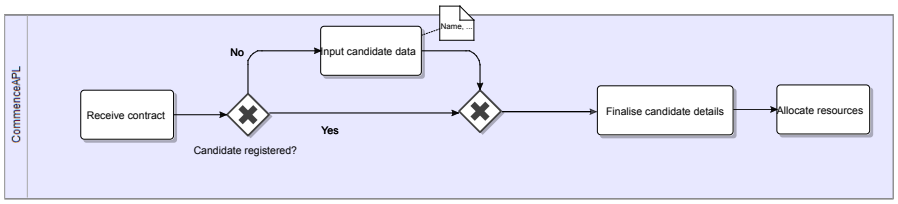


Figure A.2: BPMN Model of the APL Scenario: "Commence APL" Part (Phase)

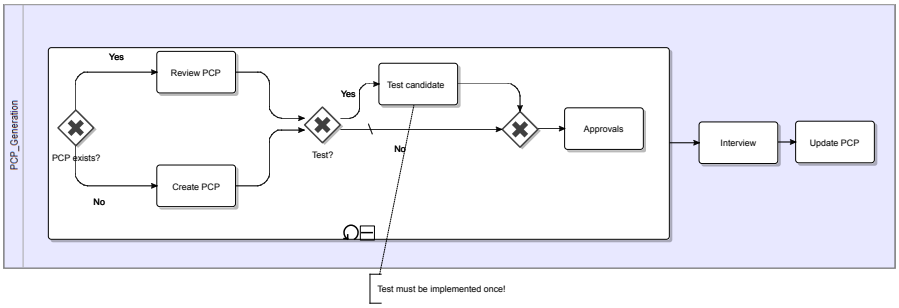


Figure A.3: BPMN Model of the APL Scenario: "PCP Generation" Part (Phase)

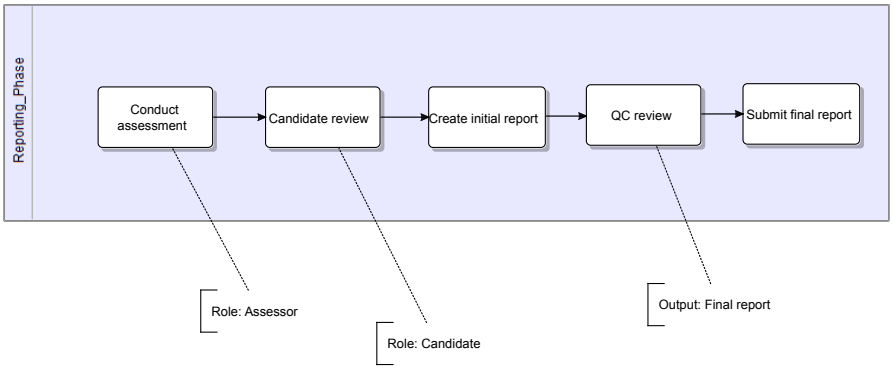


Figure A.4: BPMN Model of the APL Scenario: "Reporting" Part (Phase)

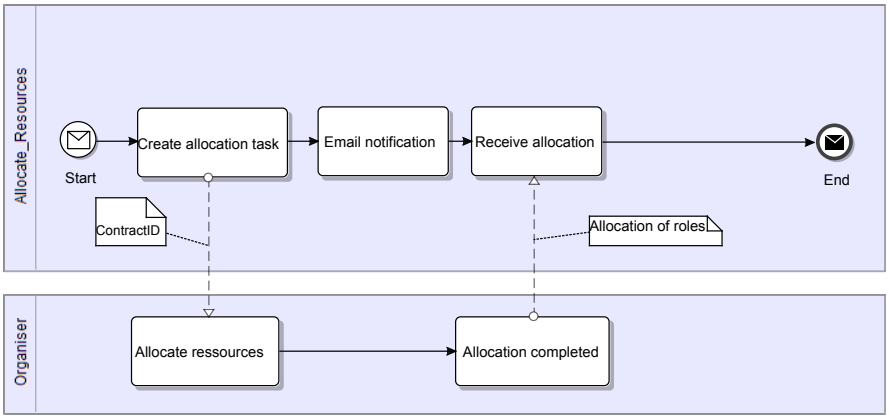


Figure A.5: BPMN Model of the APL Scenario: "Allocate Resources" Part (Scenario)

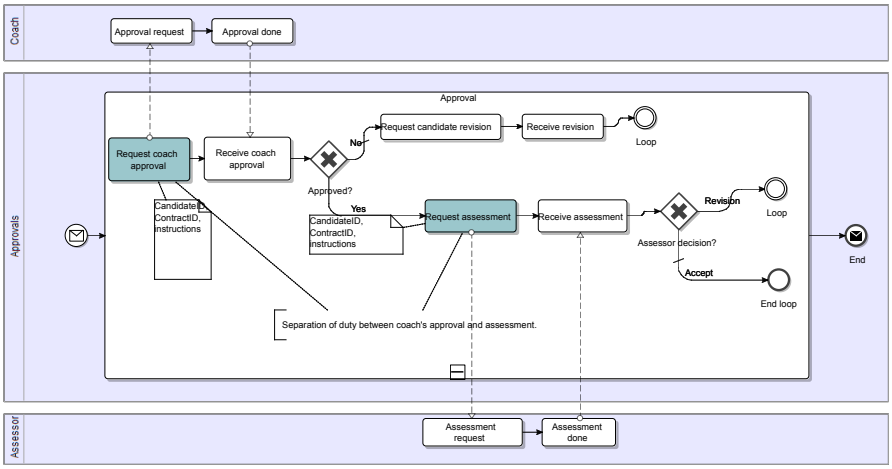


Figure A.6: BPMN Model of the APL Scenario: "Approvals" Part (Scenario)

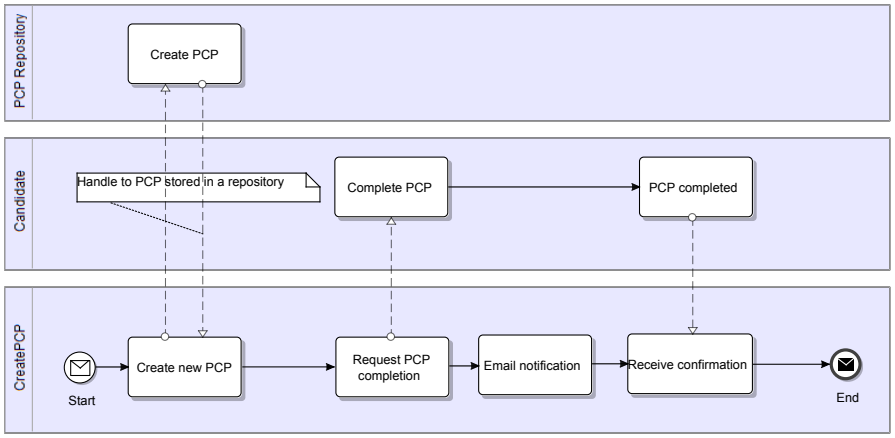


Figure A.7: BPMN Model of the APL Scenario: "Create PCP" Part (Scenario)

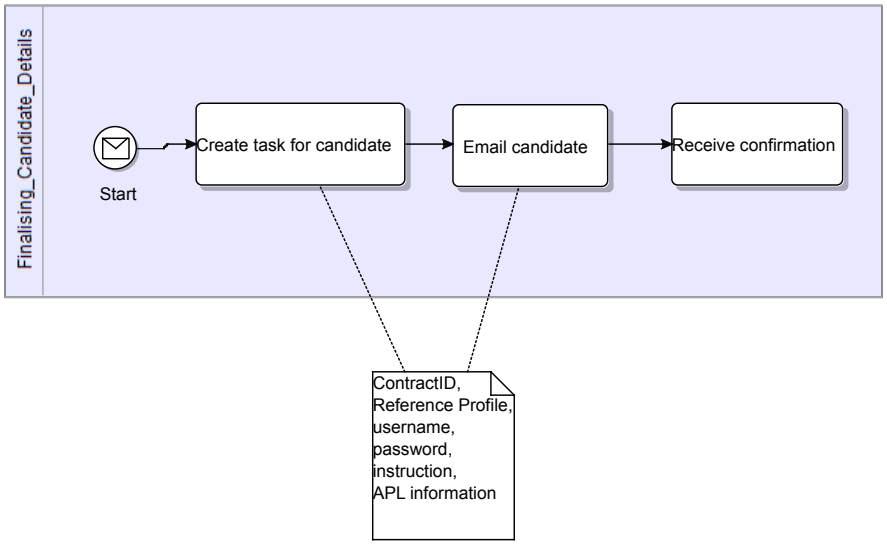


Figure A.8: BPMN Model of the APL Scenario: "Finalising Candidate Details" Part (Scenario)

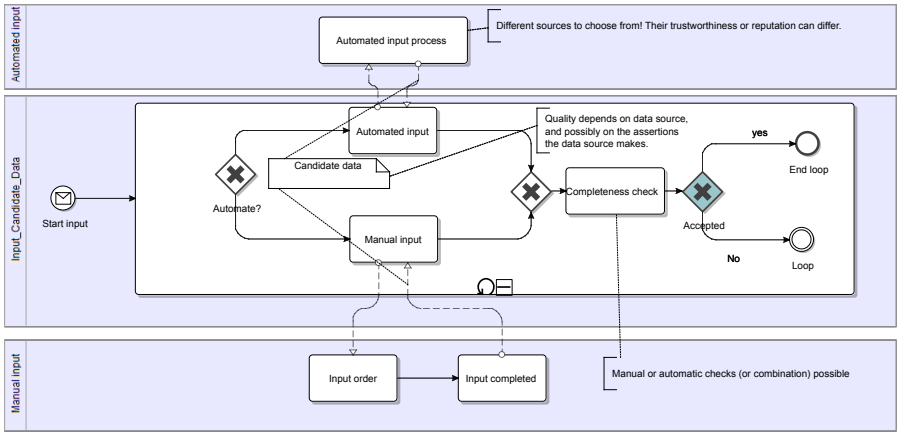


Figure A.9: BPMN Model of the APL Scenario: "Input Candidate Data" Part (Scenario)

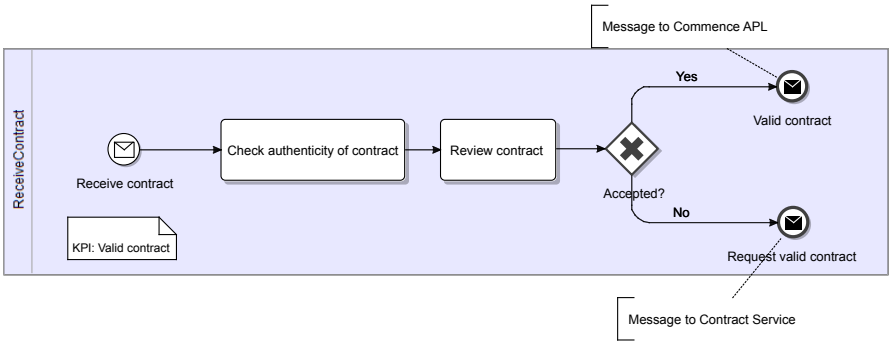


Figure A.10: BPMN Model of the APL Scenario: "Receive Contract" Part (Scenario)

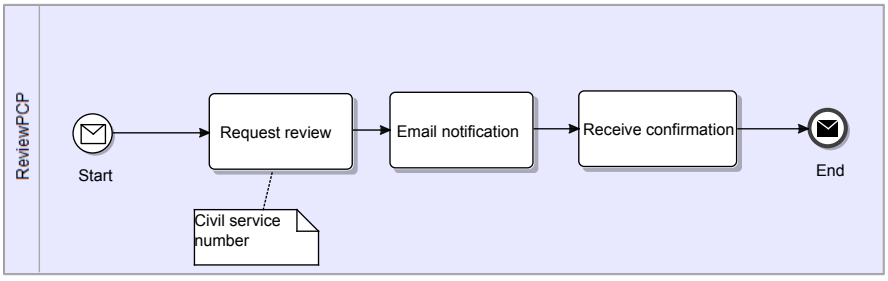


Figure A.11: BPMN Model of the APL Scenario: "Review PCP" Part (Scenario)

Appendix B

Implementation Details: Data Formats

B.1 Data model for tasks

In the field of workflow management systems, so-called human tasks usually work as follows: Some information is provided to a human user, this user performs the tasks based on the information shown, and then enters some information (in the simplest case a boolean decision), which is sent back to the workflow.

First, we can make a distinction based on the direction of the data flow. The naming (input vs. output) is derived from a view of the human task as a function.

- Input parameters: These are parameters set by the workflow and sent to the client application. The user can see the value, but not modify it.
- Output parameters: These are initially empty or set to some default value (cf. below). The user can enter a value which is sent to the workflow on completion of the task.
- Input/output parameters: This is a combination of the two types mentioned before. The workflow sets a value, which is shown to the user. The user can choose to modify this value. The final value is sent to the workflow on completion of the task.
- Finally, there is data that is not part of the *interface* of a task, but of its definition. This includes, but is not limited to, default values for output parameters, captions for parameters, text shown in the form for informational purposes, and the general formatting of the form. While this is certainly a major issue for commercial workflow systems, and only insufficiently addressed as of yet, the focus of TAS^3 is not on user interfaces, but on security. We thus want to focus on the *security* features unique to a TAS^3-embedded secure BPMS. We want to provide a clean user interface needed to demonstrate these features without wasting effort on sophisticated user interfaces that are hard to implement if they are supposed to be sufficiently general. In fact, generation of user interfaces for workflows is a research area of its own.

The second distinction is based on the data types. While there are sophisticated data type models – for instance, in XML Schema and HTML5 forms[1] – this is again not the focus of TAS[3] research. Here is an incomplete list of some of the required data types of generic importance. We list some data types needed of generic importance, but without making a claim to be complete:

- Free-form text: This type can also be used to display and enter numbers and similar. While more specific types would allow form input (n.b.: these relate to *output* parameters as introduced above) validation, free-form strings allow any data to be shown to and entered by the user. Conversion and validation then need to be performed by the workflow.

- Boolean values: This would be displayed as a check-box in front of a text caption. Such fields can be used both as input and output parameters.

- List of choices (dynamic): For this parameter type, a text caption is followed by a drop-down list. The values are set by the workflow at runtime (input parameters), as is the preselected (default) value. The user selects one of the values, and the identifier is sent back to the workflow.

- List of choices (static): This is the same as the previous item, except that the choices are static and need not be passed by the workflow.

- URL: This is a special case of a text parameter. It is only useful as an input parameter to display the URL as a clickable link. In addition to the URL itself, a link title should be given.

Note that it is not necessary to use a parameter as an output parameter only because this is stated as possible in the list above. The task definition should allow allow a parameter to be set as input-only, which will result in a rendering as read-only in the form.

Having considered the parameter types, we now look at the non-dynamic parts of the task definition. Again, we are looking for a clean and lean solution for applications to showcase security, not for a generic solution that would necessarily be very broad and complex. We deem the following items important:

- A title to be displayed as a heading.

- Free-form text to be displayed as an explanation. While allowing markup, e.g. HTML, would allow for formatting, this would require validation of the markup and a definition of allowed markup, or else would open security vulnerabilities. Thus, we opt for text-only. Any markup will be escaped and thus not rendered.

- Captions for input/output form elements. Again, this is simple text.

We have now considered all relevant aspects for designing a format for task definitions. The following is an example of a task definition in XML, covering all elements listed above:

[1] `http://dev.w3.org/html5/spec/Overview.html#states-of-the-type-attribute`

```
<taskdefinition xmlns="urn:tas3:wp3:schema:taskdef"
    id="urn:tas3:examples:taskdef1">
 <title>Compare two numbers (i.e. check whether x &lt; y)</title>
 <text>Please compare the numbers x and y given below
       and say which is larger</text>
 <input type="text" caption="x" id="number_x" readonly="true" />
 <input type="text" caption="y" id="number_y" readonly="true" />
 <input type="static_choice" caption="Which number is larger?"
        id="x_or_y">
  <!-- attribute "readonly" not given, thus considered false. -->
  <option value="x" text="x" />
  <option value="y" text="y" />
  <option value="same" text="Both are equal." />
 </input>
 <input type="boolean" caption="I think my answer is correct."
        id="correct" />
 <text>As our way to say 'thank you', we would like to offer
  you a dish of ice-cream. Please choose from the available
  flavours.</text>
 <input type="dynamic_choice" caption="Ice-cream flavour"
        id="flavour" />
   <!-- Flavours are only known at runtime and
        have to be passed by the workflow. -->
 <text>All your data is treated with due care.
       Please find a link to our privacy policy below.</text>
 <input type="url" id="privacy_policy">
  <!-- Both the caption and the URL itself have to be passed
       at runtime. -->
</taskdefinition>
```

B.2 State model for tasks

There are quite sophisticated state models for tasks, e.g. in the WS-HumanTask specification[AA+07a]. We limit our implementation to three states, as follows:

- Created: The task has been created and is ready to be performed, but is not assigned to any specific user.
- Claimed: A specific user has indicated that they are working on the task, but have not yet completed it.
- Completed: A user has performed the tasks and the results have been sent back to the workflow.

Note that the states *Claimed* and *Completed* are parameterised with a user identity.

These states are especially important for enforcing history-based constraints on authorisation for task execution. A task in the *Created* state does not affect any authorisation decision. A task *claimed* by a user may be performed by that user, so it should be treated like a completed task for access control purposes.

B.3 Database model for tasks used by the worklist handler

To store the task definition, we use the following database tables:

- A table `taskdefs`, with the following fields:
  - `id` of type String: the id of the task definition
  - `version` of type Integer: an incrementing index distinguishing different versions of task definitions with the same id. Together with the `id` field it forms the primary key for the table.
  - `title` of type String
- A table `taskdef_elements`, with the following fields:
  - `taskdef_id` of type String and `taskdef_version` of type Integer: Together, a foreign key to the `taskdefs` table.
  - `type` of type String: Either "text" or "input", corresponding to `<text>` or `<input>` elements in the task definition
  - `input_type` of type String: `null` for `<text>` elements, for `<input>` elements possible values are "text". "boolean". "static_choice". "dynamic_choice". and "url".
  - `element_id` of type String: The id of the input element where applicable, the string "null" otherwise.
  - `element_index` of type Integer: The sequential index of the corresponding `<text>` or `<input>` element. This is important for form rendering and web-service interfaces.
  - `read_only` of type Boolean: Determines whether the input element is read-only. `null` where not applicable.
  - `caption` of type String: The caption of the form element, where applicable. For `<text>` elements, the field contains the content of the element. Otherwise, a `null` value is stored.
- A table `taskdef_choices`, which lists the option of `static_choice` input elements and has the following fields:
  - `taskdef_id` of type String, `taskdef_version` of type Integer, and `element_id` of type String: Together, they form a foreign key to the `taskdef_elements` table.
  - `value` of type String: The value identifying the option.
  - `text` of type String: A description of the option, displayed to the user.
  - `element_index` of type Integer: The sequential index of the option, used for rendering.

Bibliography

[AA+07a] Active Endpoints, Adobe, BEA, IBM, Oracle, SAP AG.
Web Services Human Task (WS-HumanTask), Version 1.0, 2007.

[AA+07b] Active Endpoints, Adobe, BEA, IBM, Oracle, SAP AG.
WS-BPEL Extension for People (BPEL4People), Version 1.0, 2007.

[AH96] Vijayalakshmi Atluri and Wei-Kuang Huang.
An Authorization Model for Workflows.
In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, *Computer Security — ESORICS 96*, volume 1146 of *Lecture Notes in Computer Science*, pages 44–64. Springer Berlin / Heidelberg, 1996.
10.1007/3-540-61770-1_27.

[AH00] Vijayalakshmi Atluri and Wei-Kuang Huang.
A Petri net based safety analysis of workflow authorization models.
J. Comput. Secur., 8(2,3):209–240, August 2000.

[AL05] Anne Anderson and Hal Lockhart.
SAML 2.0 profile of XACML v2.0.
OASIS Standard, OASIS, February 2005.
http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf.

[And05] Anne Anderson.
Core and hierarchical role based access control (RBAC) profile of XACML v2.0.
Technical report, OASIS, February 2005.

[ANSI04] ANSI/INCITS.
INCITS 359-2004: Information Technology - Role Based Access Control.
Standard, February 2004.

[Ant07] Anthony Nadalin et al. (eds.).
WS-Trust 1.3.
Technical report, OASIS, March 2007.

[AS99] Gail-Joon Ahn and Ravi Sandhu.
The RSL99 Language for Role-based Separation of Duty Constraints.

In *Proceedings of the Fourth ACM Workshop on Role-based Access Control*, RBAC '99, pages 43–54, New York, NY, USA, 1999. ACM.

[AW05] Vijayalakshmi Atluri and Janice Warner.
Supporting conditional delegation in secure workflow management systems.
In *Proceedings of the tenth ACM symposium on Access control models and technologies*, SACMAT '05, pages 49–58, New York, NY, USA, 2005. ACM.

[BBK11] David Basin, Samuel J. Burri, and Gunter Karjoth.
Obstruction-Free Authorization Enforcement: Aligning Security with Business Objectives.
In *Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium*, CSF '11, pages 99–113, Washington, DC, USA, 2011. IEEE Computer Society.

[BCP06] E. Bertino, J. Crampton, and F. Paci.
Access Control and Authorization Constraints for WS-BPEL.
In *International Conference on Web Services, 2006 (ICWS '06)*, pages 275 –284, September 2006.

[BDOS05] Piero Bonatti, Claudiu Duma, Daniel Olmedilla, and Nahid Shahmehri.
An Integration of Reputation-based and Policy-based Trust Management.
In *Proceedings of the Semantic Web Policy Workshop*, 2005.

[BEdH+10] Klemens Böhm, Sandro Etalle, Jerry den Hartog, Christian Hütter, Slim Trabelsi, Daniel Trivellato, and Nicola Zannone.
A Flexible Architecture for Privacy-Aware Trust Management.
Journal of Theoretical and Applied Electronic Commerce Research, 5(2):77–96, 2010.

[BK12] Samuel J. Burri and Günter Karjoth.
Flexible Scoping of Authorization Constraints on Business Processes with Loops and Parallelism.
In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 100 of *Lecture Notes in Business Information Processing*, pages 411–422. Springer Berlin Heidelberg, 2012.

[BMPS09] Elisa Bertino, Lorenzo Martino, Federica Paci, and Anna Squicciarini.
Security for Web Services and Service-Oriented Architectures.
Springer, Berlin/Heidelberg, 2009.

[BPM+08] Tim Bray, Jean Paoli, Eve Maler, François Yergeau, and C. M. Sperberg-McQueen.
Extensible Markup Language (XML) 1.0 (Fifth Edition).
W3C Recommendation, W3C, November 2008.
http://www.w3.org/TR/2008/REC-xml-20081126/.

[BS00] E. Barka and R. Sandhu.
Framework for role-based delegation models.

In *Proceedings of the 16th Annual Computer Security Applications Conference*, ACSAC '00, pages 168–176, Washington, DC, USA, 2000. IEEE Computer Society.

[BS04] E. Barka and Ravi Sandhu.
Role-based Delegation Model/Hierarchical Roles (RBDM1).
In *Computer Security Applications Conference, 2004. 20th Annual*, pages 396 – 404, dec. 2004.

[BTT+09] Tim Bray, Richard Tobin, Henry S. Thompson, Dave Hollander, and Andrew Layman.
Namespaces in XML 1.0 (Third Edition).
W3C Recommendation, W3C, December 2009.
http://www.w3.org/TR/2009/REC-xml-names-20091208/.

[Cam06] Kim Cameron.
The Laws of Identity.
www.identityblog.com/?p=352, 2006.
Last accessed 16 March 2014.

[CC07] Jeff Hodges Conor Cahill.
Liberty ID-WSF Discovery Service Specification.
Specification, Liberty Alliance, 2007.
http://projectliberty.org/liberty/content/download/3449/22973/file/liberty-idwsf-disco-svc-2.0-errata-v1.0.pdf.

[CCMe01] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana (eds.).
Web Services Description Language (WSDL) 1.1.
W3C Note, W3C, March 2001.
http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[CF09] David W. Chadwick and Kaniz Fatema.
An advanced policy based authorisation infrastructure.
In *Proceedings of the 5th ACM workshop on Digital identity management*, DIM 2009, pages 81–84, 2009.

[CF12] David W. Chadwick and Kaniz Fatema.
A privacy preserving authorisation system for the cloud.
J. Comput. Syst. Sci., 78(5):1359–1373, September 2012.

[Cha07] Anis Charfi.
Aspect-Oriented Workflow Languages: AO4BPEL and Applications.
PhD thesis, Technische Universität Darmstadt, Fachbereich Informatik, 2007.

[Cha08] David W. Chadwick.
Securing Web Services: Practical Usage of Standards and Specifications, chapter Dynamic Delegation of Authority in Web Services.
IGI Global, 2008.

[Cha09] David Chadwick.
Federated Identity Management.
In Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design V*, volume 5705

of *Lecture Notes in Computer Science*, pages 96–120. Springer Berlin / Heidelberg, 2009.

[CHe06] Roberto Chinnici, Marc Hadley, and Rajiv Mordani (eds.).
The Java API for XML-Based Web Services (JAX-WS) 2.0.
JCP Specification JSR-224, Final Release, Java Community Process, April 2006.
http://jcp.org/aboutJava/communityprocess/final/jsr224/index.html.

[CKO92] Bill Curtis, Marc I. Kellner, and Jim Over.
Process modeling.
Communications of the ACM, 35:75–90, September 1992.

[CKPM05] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler.
Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.
Oasis standard, OASIS, March 2005.
http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[CM06] Anis Charfi and Mira Mezini.
Aspect-Oriented Workflow Languages.
In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *Lecture Notes in Computer Science*, pages 183–200. Springer Berlin / Heidelberg, 2006.

[CWK09] Brecht Claerhout, Sandra Winfield, and Tom Kirkham.
Pilot specification and use case scenarios.
Project Delieverable D9.1 (version 3.0 Resubmitted), TAS3 Consortium, December 2009.
http://www.tas3.eu/.

[CWMe07] Roberto Chinnici, Sanjiva Weerawarana, Jean-Jacques Moreau, and Arthur Ryman (eds.).
Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.
W3C recommendation, W3C, June 2007.
http://www.w3.org/TR/2007/REC-wsdl20-20070626.

[CXO+07] David W. Chadwick, Wensheng Xu, Sassa Otenko, Romain Laborde, and Bassem Nasser.
Multi-session Separation of Duties (MSoD) for RBAC.
In *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop*, ICDEW '07, pages 744–753, Washington, DC, USA, 2007. IEEE Computer Society.

[Dey01] Anind K. Dey.
Understanding and Using Context.
Personal Ubiquitous Comput., 5(1):4–7, January 2001.

[Erl05] Thomas Erl.
Service-Oriented Architecture: Concepts, Technology, and Design.
Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[EC95] European Community.

Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data.

[FK92] David Ferraiolo and Richard Kuhn.
Role-based Access Controls.
In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[GD07] Rick Geneva and Tom Debevoise.
The Microguide to Process Modeling in BPMN.
BookSurge Publishing, July 2007.

[GGF98] V.D. Gligor, S.I. Gavrila, and D. Ferraiolo.
On the formal definition of separation-of-duty policies and their composition.
In *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*, pages 172 –183, May 1998.

[GHM+07] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Yves Lafon, Jean-Jacques Moreau, Anish Karmarkar, and Henrik Frystyk Nielsen.
SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).
W3C Recommendation, W3C, April 2007.
http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[GMPT01] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos, and Roshan K. Thomas.
Flexible Team-based Access Control Using Contexts.
In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, SACMAT '01, pages 21–27, New York, NY, USA, 2001. ACM.

[Gör09] Heiko Görig.
Kontextbezogene Zugriffskontrolle für BPEL (Engines).
Diplomarbeit, Universität Stuttgart, April 2009.

[HAMC07] Jeff Hodges, Robert Aarts, Paul Madsen, and Scott Cantor.
Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification.
Technical report, Liberty Alliance, 2007.
http://projectliberty.org/liberty/content/download/3439/22943/file/liberty-idwsf-authn-svc-2.0-errata-v1.0.pdf.

[HB08] Michael Hafner and Ruth Breu.
Security Engineering for Service-Oriented Architectures.
Springer Publishing Company, Incorporated, 1 edition, 2008.

[HCH+05] John Hughes, Scott Cantor, Jeff Hodges, Frederick Hirsch, Prateek Mishra, Rob Philpott, and Eve Maler.
Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0.
OASIS Standard, OASIS, March 2005.

http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.

[HGS+11] Waldemar Hummer, Patrick Gaubatz, Mark Strembeck, Uwe Zdun, and Schahram Dustdar.
An Integrated Approach for Identity and Access Management in a SOA Context.
In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies*, SACMAT '11, pages 21–30, New York, NY, USA, 2011. ACM.

[Hir06a] Frederick Hirsch.
ID-WSF 2.0 SecMech SAML Profile.
Technical report, Liberty Alliance, 2006.
http://projectliberty.org/liberty/content/download/894/6258/file/liberty-idwsf-security-mechanisms-saml-profile-v2.0.pdf.

[Hir06b] Frederick Hirsch.
Liberty ID-WSF Security Mechanisms Core.
Technical report, Liberty Alliance, 2006.
http://projectliberty.org/liberty/content/download/3478/23060/file/liberty-idwsf-security-mechanisms-core-2.0-errata-v1.0.pdf.

[HKA+07] Jeff Hodges, John Kemp, Robert Aarts, Greg Whitehead, and Paul Madsen.
Liberty ID-WSF SOAP Binding Specification.
Technical report, Liberty Alliance, 2007.
http://www.projectliberty.org/liberty/content/download/3483/23075/file/liberty-idwsf-soap-binding-2.0-errata-v1.0.pdf.

[HL10] E. Hammer-Lahav.
The OAuth 1.0 Protocol.
RFC 5849 (Informational), April 2010.

[Hol95] David Hollingsworth.
The Workflow Reference Model.
WfMC Specification TC00-1003, Workflow Management Coalition, 1995.

[ITU91] International Telecommunications Union.
Security Architecture for Open Systems Interconnection for CCITT applications (Recommendation X.800), March 1991.

[ITU95a] International Telecommunications Union.
Security Frameworks for Open Systems: Access Control Framework (ITU-T Recommendation X.812), November 1995.

[ITU95b] International Telecommunications Union.
Security Frameworks for Open Systems: Integrity Frameworks (ITU-T Recommendation X.815), November 1995.

[ITU95c] International Telecommunications Union.
Security Frameworks for Open Systems: Overview (ITU-T Recommendation X.810), November 1995.

[ISS] ISSRG: University of Kent, Information Systems Security Research Group.
Modular PERMIS Project.

[JB96] Stefan Jablonski and Christoph Bussler.
Workflow Management – Modeling Concepts, Architecture and Implementation.
International Thomson Computer Press, 1996.

[JE07] Diane Jordan and John Evdemon.
Web Services Business Process Execution Language Version 2.0.
OASIS Standard, OASIS, April 2007.
http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[JT06] Yuzo Koga Jonathan Tourzan.
Liberty ID-WSF Web Services Framework Overview, Version 2.0.
Technical report, Liberty Alliance, 2006.
http://projectliberty.org/liberty/content/download/889/6243/file/liberty-idwsf-overview-v2.0.pdf.

[KC08] Ramarao Kanneganti and Prasad Chodavarapu.
SOA Security.
Manning Publications Co., Greenwich, Connecticut, USA, 2008.

[KCM+05] John Kemp, Scott Cantor, Prateek Mishra, Rob Philpott, and Eve Maler.
Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0.
OASIS Standard, OASIS, March 2005.
http://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf.

[KF95] D. Richard Kuhn and David F. Ferraiolo.
Role-Based Access Control (RBAC): Features and Motivations.
Proceedings of the 11th Annual Computer Security Application Conference, pages 241–248, 1995.

[KNS05] Karl Krukow, Mogens Nielsen, and Vladimiro Sassone.
A Framework for Concrete Reputation-systems with Applications to History-based Access Control.
In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, pages 260–269, 2005.

[Lan03] Susan Landau.
Liberty ID-WSF Security and Privacy Overview.
Technical report, Liberty Alliance, 2003.
http://www.projectliberty.org/liberty/content/download/1298/8259/file/liberty-idwsf-security-privacy-overview-v1.0.pdf.

[Le07] Canyang Kevin Liu and David Booth (eds.).
Web Services Description Language (WSDL) Version 2.0 Part 0: Primer.
W3C Recommendation, W3C, June 2007.
http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626.

[LHC+07] Amelia A. Lewis, Hugo Haas, Roberto Chinnici, Sanjiva Weerawarana, Jean-Jacques Moreau, and David Orchard.
Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts.
W3C Recommendation, W3C, June 2007.

http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626.

[Lib06] Liberty Alliance Project.
Liberty ID-WSF Web Services Framework Overview Version 2.0, 2006.

[LK06a] Kelvin Lawrence and Chris Kaler.
Web Services Security: SAML Token Profile 1.1.
OASIS Standard, OASIS, February 2006.
http://docs.oasis-open.org/wss/oasis-wss-SAMLTokenProfile-1.1.

[LK06b] Kelvin Lawrence and Chris Kaler.
Web Services Security: SOAP Message Security 1.1 (WS-Security 2004).
OASIS Standard Specification, OASIS, February 2006.
http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[MB04] Ashok Malhotra and Paul V. Biron.
XML Schema Part 2: Datatypes (Second Edition).
W3C Recommendation, W3C, October 2004.
http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[MB11] Jens Müller and Klemens Böhm.
The Architecture of a Secure Business-Process-Management System in Service-Oriented Environments.
In *Proceedings of the 2011 Ninth IEEE European Conference on Web Services*, pages 49–56, 2011.

[MB14] Jens Müller and Klemens Böhm.
Identity Business Processes.
Int. J. of Trust Management in Computing and Communications, 2(1):40–77, 2014.

[MGH+07] Jean-Jacques Moreau, Martin Gudgin, Marc Hadley, Noah Mendelsohn, Yves Lafon, Anish Karmarkar, and Henrik Frystyk Nielsen (eds.).
SOAP Version 1.2 Part 2: Adjuncts (Second Edition).
W3C Recommendation, W3C, April 2007.
http://www.w3.org/TR/2007/REC-soap12-part2-20070427/.

[MKB12a] Jens Müller, Murat Kavak, and Klemens Böhm.
A Graphical Audit Facility for Data Processing and its Evaluation with Users.
Technical Report 2012, 1, Karlsruhe Reports in Informatics, 2012.

[MKB12b] Jens Müller, Murat Kavak, and Klemens Böhm.
A Graphical Audit Facility for Data Processing and Its Evaluation with Users.
In Quan Z. Sheng, Guoren Wang, Christian S. Jensen, and Guandong Xu, editors, *Web Technologies and Applications*, volume 7235 of *Lecture Notes in Computer Science*, pages 618–627. Springer Berlin Heidelberg, 2012.

[ML07] Nilo Mitra and Yves Lafon.
SOAP Version 1.2 Part 0: Primer (Second Edition).
Technical report, W3C, April 2007.

http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

[MMvSB10] Jens Müller, Jutta Mülle, Silvia von Stackelberg, and Klemens Böhm.
Secure Business Processes in Service-Oriented Architectures – A Requirements Analysis.
In *Proceedings of the 2010 Eighth IEEE European Conference on Web Services*, ECOWS '10, pages 35–42, Washington, DC, USA, 2010. IEEE Computer Society.

[Mos05] Tim Moses.
eXtensible Access Control Markup Language (XACML) Version 2.0.
OASIS Standard, OASIS, February 2005.
http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[MPS08] Jan Mendling, Karsten Ploesser, and Mark Strembeck.
Specifying Separation of Duty Constraints in BPEL4People Processes.
In *Business Information Systems*, volume 7, pages 273–284. 2008.

[MR08] Eve Maler and Drummond Reed.
The Venn of Identity: Options and Issues in Federated Identity Management.
IEEE Security and Privacy, 6(2):16–23, March 2008.

[MTM09] Michael Menzel, Ivonne Thomas, and Christoph Meinel.
Security Requirements Specification in Service-Oriented Business Process Management.
In *Availability, Reliability and Security, International Conference on*, pages 41–48, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[MvSB11a] Jutta Mülle, Silvia von Stackelberg, and Klemens Böhm.
A Security Language for BPMN Process Models.
Technical Report 2011, 9, Karlsruhe Reports in Informatics, 2011.

[MvSB11b] Jutta Mülle, Silvia von Stackelberg, and Klemens Böhm.
Modelling and Transforming Security Constraints in Privacy-Aware Business Processes.
In *Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications*, 2011.

[NGG+07] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist.
WS-SecurityPolicy 1.2.
OASIS Standard, OASIS, July 2007.
http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html.

[OAS08] OASIS.
Security Assertion Markup Language (SAML) V2.0 Technical Overview, 2008.

[Obj08] Object Management Group.
Business Process Modeling Notation, Version 1.1.

OMG Available Specification, January 2008.

[Obj11] Object Management Group.
Business Process Modeling Notation, Version 2.0.
OMG Available Specification, January 2011.

[OEC13] OECD (Organization for Economic Co-operation and Development).
Recommendation of the Council concerning Guidelines governing the Protection of Privacy and Transborder Flows of Personal Data (2013).
Technical Report C(80)58/FINAL, as amended on 11 July 2013 by C(2013)79, July 2013.

[Ope07] OpenID Foundation.
OpenID Authentication 2.0, 2007.

[Par72] D. L. Parnas.
On the Criteria To Be Used in Decomposing Systems into Modules.
Communications of the ACM, 15:1053–1058, December 1972.

[PFB09] Federica Paci, Rodolfo Ferrini, and Elisa Bertino.
Identity Attribute-Based Role Provisioning for Human WS-BPEL Processes.
In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 535–542, Washington, DC, USA, 2009. IEEE Computer Society.

[PH10] Andreas Pfitzmann and Marit Hansen.
A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, and Identity Management (v0.34).
Technical report, August 2010.
http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf.

[PP02] Charles P. Pfleeger and Shari Lawrence Pfleeger.
Security in Computing.
Prentice Hall Professional Technical Reference, 3rd edition, 2002.

[Res00] E. Rescorla.
HTTP Over TLS.
RFC 2818 (Informational), May 2000.
Updated by RFC 5785.

[RFMP07] Alfonso Rodríguez, Eduardo Fernández-Medina, and Mario Piattini.
A BPMN Extension for the Modeling of Security Requirements in Business Processes.
IEICE - Trans. Inf. Syst., E90-D(4):745–752, March 2007.

[Ris13] Erik Rissanen.
eXtensible Access Control Markup Language (XACML) Version 3.0.
OASIS Standard, OASIS, January 2013.
http://docs.oasis-open.org/xacml/xacml-saml-profile/v2.0/cs02/xacml-saml-profile-v2.0-cs02.pdf.

[Ris14a] Erik Rissanen.
XACML SAML profile version 2.0.
Committee Specification 02, OASIS, August 2014.
http://docs.oasis-open.org/xacml/xacml-saml-profile/v2.0/cs02/xacml-saml-profile-v2.0-cs02.pdf.

[Ris14b] Erik Rissanen.
XACML v3.0 Core and hierarchical role based access control (RBAC) profile version 1.0.
Committee Specification 02, OASIS, October 2014.

[RL14] Erik Rissanen and Hal Lockhart.
XACML v3.0 Administration and Delegation Profile Version 1.0.
Committee Specification Draft 04 / Public Review Draft 02, OASIS, November 2014.
http://docs.oasis-open.org/xacml/3.0/administration/v1.0/csprd02/xacml-3.0-administration-v1.0-csprd02.pdf.

[RRF08] Michael Rosemann, Jan Recker, and Christian Flender.
Contextualisation of Business Processes.
International Journal of Business Process Integration and Management, 3(1):47–60, 2008.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman.
A method for obtaining digital signatures and public-key cryptosystems.
Communications of the ACM, 21(2):120–126, 1978.

[Run08] Mary Rundle.
At a Crossroads: "Personhood" and Digital Identity in the Information Society.
STI Working Paper 2007/7, OECD Directorate for Science, Technology and Industry, February 2008.

[Sch00] Bruce Schneier.
Secrets and Lies – Digital Security in a Networked World.
Wiley Computer Publishing, 2000.

[SW65] S. S. Shapiro and M. B. Wilk.
An Analysis of Variance Test for Normality (Complete Samples).
Biometrika, 3(52):1–22, 1965.

[TAS09a] Design Requirements of an Adjusted, Process Driven Trust and Security Environment.
Project Delieverable D1.4 (3.0), TAS3 Consortium, May 2009.
http://www.tas3.eu/.

[TAS09b] Specification of Secure Data Repositories and Authoritative Sources.
Technical Report D4.2 (1.2), TAS3 Consortium, May 2009.
http://www.tas3.eu/.

[TAS11] Open Source Software and Documentation Implementing the Design.
Project Delieverable D3.2 (3.0), TAS3 Consortium, December 2011.
http://www.tas3.eu/.

[TCG04] Kaijun Tan, Jason Crampton, and Carl A. Gunter.
The Consistency of Task-Based Authorization Constraints in Workflow Systems.
In *Proceedings of the 17th IEEE workshop on Computer Security Foundations*, pages 155–, Washington, DC, USA, 2004. IEEE Computer Society.

[Tho97] Roshan K. Thomas.
Team-based access control (TMAC): a primitive for applying role-based access controls in collaborative environments.
In *Proceedings of the Second ACM Workshop on Role-Based Access Control*, RBAC '97, pages 13–19, New York, NY, USA, 1997. ACM.

[TMBM04] Henry S. Thompson, Murray Maloney, David Beech, and Noah Mendelsohn.
XML Schema Part 1: Structures (Second Edition).
W3C Recommendation, W3C, October 2004.
http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[TS98] Roshan K. Thomas and Ravi S. Sandhu.
Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Autorization Management.
In *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI: Status and Prospects*, pages 166–181, London, UK, UK, 1998. Chapman & Hall, Ltd.

[Ved07] Asir S. Vedamuthu.
Web Services Description Language (WSDL) Version 2.0 SOAP 1.1 Binding.
W3C Note, W3C, June 2007.
http://www.w3.org/TR/2007/NOTE-wsdl20-soap11-binding-20070626.

[VYO+07] Asir S Vedamuthu, Ümit Yalçinalp, David Orchard, Toufic Boubez, Frederick Hirsch, Prasad Yendluri, and Maryann Hondo.
Web Services Policy 1.5 - Attachment.
W3C Recommendation, W3C, September 2007.
http://www.w3.org/TR/2007/REC-ws-policy-attach-20070904.

[WBK03] Jacques Wainer, Paulo Barthelmess, and Akhil Kumar.
W-RBAC - A workflow security model incorporating controlled overriding of constraints.
Int. J. Cooperative Inf. Syst., 12(4):455–485, 2003.

[Wes07] Mathias Weske.
Business Process Management: Concepts, Languages, Architectures.
Springer, 2007.

[WF04] Priscilla Walmsley and David C. Fallside.
XML Schema Part 0: Primer (Second Edition).
W3C Recommendation, W3C, October 2004.
http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

[WfMC98] Workflow Management Coalition.
Audit Data Specification, 1998.

[WK05] Jacques Wainer and Akhil Kumar.
A fine-grained, controllable, user-to-user delegation method in RBAC.
In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, SACMAT '05, pages 59–66, New York, NY, USA, 2005. ACM.

[WM07] Michael E. Whitman and Herbert J. Mattord.
Principles of Information Security.
Course Technology Press, Boston, MA, United States, 3rd edition, 2007.

[WMS+09] Christian Wolter, Michael Menzel, Andreas Schaad, Philip Miseldine, and Christoph Meinel.
Model-Driven Business Process Security Requirement Specification.
Journal of Systems Architecture, 55(4):211 – 223, 2009.

[WS07] Christian Wolter and Andreas Schaad.
Modeling of task-based authorization constraints in BPMN.
In *Proceedings of the 5th international conference on Business process management*, BPM'07, pages 64–79, Berlin, Heidelberg, 2007. Springer-Verlag.

[WSM07] Christian Wolter, Andreas Schaad, and Christoph Meinel.
Deriving XACML Policies from Business Process Models.
In *Proceedings of the 2007 International Conference on Web Information Systems Engineering*, WISE'07, pages 142–153, Berlin, Heidelberg, 2007. Springer-Verlag.

[YCW+10] Jinhui Yao, Shiping Chen, Chen Wang, David Levy, and John Zic.
Accountability as a Service for the Cloud.
In *Proceedings of the 2010 IEEE International Conference on Services Computing*, pages 81–88, 2010.

[YHV+07] Prasad Yendluri, Maryann Hondo, Asir S Vedamuthu, Frederick Hirsch, David Orchard, Ümit Yalçinalp, and Toufic Boubez.
Web Services Policy 1.5 - Framework.
W3C Recommendation, W3C, September 2007.
http://www.w3.org/TR/2007/REC-ws-policy-20070904.

[ZAC03] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu.
A Rule-Based Framework for Role-Based Delegation and Revocation.
ACM Transactions on Information and System Security, 6(3):404–441, August 2003.

[zM08] zur Muehlen, M. (ed.).
Business Process Analytics Format (BPAF).
WfMC Draft Standard WFMC-TC-1015, February 2008.

List of Tables

List of Figures

Glossary

ABAC Attribute-Based Access Control 19
AOP Aspect-Oriented Programming 33
APL Accreditation of Prior Learning 39

BPCL Business-Process Constraint Language 32
BPEL Business Process Execution Language 15
BPMN Business Process Model and Notation 15
BPMN Business Process Modelling Notation 14

CCITT *Comité Consultatif International Téléphonique et Télégraphique* / International Telegraph and Telephone Consultative Committee 17

DAC Discretionary Access Control 18
DSL Domain-specific Language 31

ECTS European Credit Transfer and Accumulation System 1

FIM Federated Identity Management 21

ID-WSF Identity Web Services Framework 23
IdP Identity Provider 22
ITU International Telecommunications Union 17
ITU-T ITU Telecommunication Standardization Sector 17

MSoD Multi-session Separation of Duty 19

OECD Organisation for Economic Co-operation and Development 24

PAP-Wf Policy Administration Point for Workflows 81
PCP Personal Competency Profile 44
PEP-HT Policy Enforcement Point for Human Tasks 79
PEP-WS Policy Enforcement Point for Web Services 78
PIP Policy Information Point 79
PMF Process Modelling Framework 40

RBAC Role-based Access Control 19

SAML Security Assertion Markup Language 21
SOA Service-oriented architecture 7

SoD Separation of Duty 19
SSO Single Sign-On 21

TAS^3 Trusted Architecture for Securely Shared Services 39
TBAC Task-based Authorization Controls 28
TMAC Team-Based Access Control 27

WAM Workflow Authorization Model 28
WfMC Workflow Management Coalition 15
WfMS Workflow Management System 12
WS-BPEL Web Services Business Process Execution Language 15
WSDL Web Services Description Language 11

XACML Extensible Access Control Markup Language 20
XML eXtensible Markup Language 9