PROVIDING NATIVE SUPPORT FOR FEDERATED IDENTITY MANAGEMENT IN A BUSINESS-PROCESS-MANAGEMENT SYSTEM

Identity Business Processes

Jens Müller

Faculty of Informatics,
Karlsruhe Institute of Technology,
Karlsruhe, Germany
E-mail: jens.mueller@kit.edu

Klemens Böhm

Faculty of Informatics,
Karlsruhe Institute of Technology,
Karlsruhe, Germany
E-mail: jens.mueller@kit.edu

Abstract: To facilitate information-system security, e. g., access control or audit, the entities involved play a key role. This makes *identity management* an important task. The success of service-oriented architectures (SOA) has lead to the development of *federated identity management* (FIM), to deal with the dynamic nature of SOA and to achieve economies of scale. Business processes in SOA are a composition of services provided by IT systems and manual actions performed by humans. Such compositions highly depend on the identity of participants. The identity in turn determines aspects such as preferred services or assignment of tasks. We analyze how to use FIM technologies to facilitate such identity-based compositions and identify the problems arising from this combination (business processes and FIM). Based on standards for business-process management, access control, and FIM, by carefully considerating design alternatives, we propose a system architecture for the execution of identity-based business processes. The system implements FIM concepts in an application-specific way, based on declarative configuration and taking the run-time context of business processes into account. Finally, we describe our implementation of the architecture based on the ZXID open-source library and its evaluation using a case study.

Keywords: Business-process management; access control; identity management; Service-oriented architecture (SOA).

Reference to this paper should be made as follows: Müller, J. and Böhm, K. (2014) 'Identity Business Processes: Providing Native Support for Federated Identity Management in a Business-Process-Management System', *International Journal of Trust Management in Computing and Communications*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: The research of Jens Müller deals with security mechanisms for workflow systems. He has worked in TAS^3, an EU-funded research project dealing with security and trust in service-oriented architectures and is an experienced professional software developer. He has published several conference papers and is currently working towards his PhD.

Klemens Böhm is professor for computer science (chair of databases and information systems) at Karlsruhe Institute of Technology (KIT), Germany. Prior to that, he has been affiliated with Otto-von-Guericke-Universität Magdeburg, Germany, Swiss Federal Institute of Technology Zurich, Switzerland, and GMD Darmstadt, Germany. He has obtained his PhD from the Technical University of Darmstadt in 1997. The research topics currently addressed by his chair include data mining, data privacy, and workflow management. The range of applicants he collaborates with is broad and includes engineers, biologists, and economists. Collaboration with industry also plays an important role.

1 Introduction

Service-oriented architectures (SOA) build upon the notion of services, i.e., bundles of operations that are available through clearly defined interfaces. Today, implementations of SOA commonly use web-service technologies such as SOAP [45], WSDL [13], and other technologies building upon them [14].

Business-process management has the goal of *orchestrating* high-level interactions in SOA. This means that processes manage how services interact, touching aspects such as control flow and data flow. As an example, think of a corporate travel-booking application. It combines services for booking a flight, a hotel, and a rental car. A clerk is involved to check whether the booking intended confirms to corporate guidelines for travel expenses.

Security is an important issue in such applications. For example, transmission of sensitive personal data to external services must not lead to disclosure of such data. Moreover, one must be able to trust the service in that data is secure there. Users may not perform tasks in the process without authorization. In the travel booking example, employees are allowed to perform a booking, while interns are not. Further, a clerk may not confirm his own booking, a constraint known as separation of duty (SoD) [21].

An important prerequisite to achieve these 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" [41]. *Digital identity* 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* (IdM) means managing these various partial identities.

IdM is important for security, for authentication and access control in particular. An IdM 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 some resource. In our example, an attribute `employment_status` determines whether an individual may use the travel-booking application. Authentic information about individuals is used when booking tickets.

In the SOA context, IdM has evolved into *federated identity management* (FIM), a set of technologies and processes that let computer systems distribute identity information dynamically and delegate IdM functionality to other systems [27]. A 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 [8]. The Security Assertion Markup Language (SAML) [38] implements FIM concepts. ID-WSF [24] uses SAML and related technologies to facilitate identity-aware service compositions. In our

example, the airline can automatically credit bonus miles to the traveler's frequent-flyer card of a partner airline. The booking application can enter itinerary data into the calendar of the user.

We summarize the current status of FIM as follows: Terminology and concepts have been established, and requirements on FIM have been explored. There are implementations of FIM concepts, e. g., based on SAML, and even a framework to use FIM in service compositions. However, there is no comprehensive solution for problems commonly occuring in such compositions, for example managing the different security preferences of the individuals involved. This would help making applications more user-friendly: E.g., a user might have specified which data he is willing to disclose to applications of a specific kind. When an application can access such preferences, it can avoid bothering the user unnecessarily. Moreover, service compositions are commonly implemented using business-process-management technologies, but there are no mature solutions to integrate FIM.

Integrating FIM support into a BP-management system (BPMS) facilitates what we call *identity business processes*, allowing developers of BP applications to easily use the services available in an IdM federation. To customize the FIM-related behavior of a service composition, only lightweight configuration shall be necessary. Application developers should be able to provide it separately from the BP definition itself, e. g., by annotating the process model, as proposed in [44, 28, 42] For example, to use FIM for access control, application developers need to state the attribute values required for users to perform a certain activity of the process. Given this specification, users can use SSO to log into the user interface, and the BPMS automatically authenticates the attributes of the user. Further, users have a more consistent experience when using different service providers: They can use their existing accounts at an identity provider and provide access to their personal data to applications automatically. In addition, applications can respect privacy preferences automatically.

The research question now is how to design a BPMS for identity BPs. Problems that need to be solved are: How to adapt and extend the conventional architecture of a BPMS to support FIM concepts? How can one implement advanced BPMS functionality in SOA, such as dynamic service compositions, based on FIM? Which configuration options are needed to customize the FIM-related aspects of BP definitions?

All these problems are challenging. First, it is unclear how FIM fits into the architecture of a secure BPMS. There exists a well-defined BPMS reference architecture [17]. This reference architecture is valid in a SOA context as well [16]. It provides a good basis for the integration of new functionality into a BPMS. One question now is which interfaces of this architecture are affected, i. e., have to interact with the FIM system or handle identity information? Which components of the BPMS use such information? Second, the resulting BPMS should re-use existing BPMS components, interoperate with established FIM technologies, allow execution of existing process definitions, and require little additional configuration. Third, the implementation of FIM features needs to take the process context into account and adjust to the requirements of individual process definitions. One example would be a travel-booking application involving a traveler and a clerk. After the booking is completed, the traveler agrees to store itinerary data in his calendar. The BPMS needs to discover the correct calendar service and call it with the correct identity. This depends on process context, in a way that is specific to each application. In this example, the traveler who has initiated the booking determines which calendar service to use, and the service is called with his identity. Fourth, the resulting BPMS must maintain user privacy. To this end, it must support the respective FIM features, like identity mapping between pseudonyms,

prevent leakage of identity information between process instances, and provide appropriate user control over the disclosure of their personal data.

Our contributions now are as follows:

- We determine how FIM concepts can be used in BPs. We do so by analyzing the flow of identity information and the influence of BP context on FIM functionality. We also describe peculiarities that arise when combining FIM concepts with BPM concepts.
- We analyze how FIM functionality needs to be configured so that it can meet the requirements of different BP applications, and we develop respective configuration mechanisms.
- In order to enable secure process execution in SOA with FIM, we identify important design decisions and different solutions for each of them. We discuss their pros and cons based on a number of design requirements. The result is an extended BPMS architecture.
- We describe an implementation of the architecture based on web-service technologies and the open-source ZXID library, an implementation of the SAML and ID-WSF specifications. Finally, we demonstrate how the system executes a BP implementing the travel-booking scenario from above. We also assess the architecture with respect to the design requirements.

2 Fundamentals

The goal of this article is to facilitate the implementation of identity-based service compositions in service-oriented architectures. The travel-booking use case serves as an example for such a service composition. The question now is how existing concepts and technologies can help with its implementation. This section features two major topics that are useful to this end: (1) Business-process management systems orchestrate different services, yielding more complex applications. They can also accomodate human activities. This allows to orchestrate, say, a flight-booking system and the work of a manager responsible for business trips. (2) FIM makes it easier to build service compositions that adapt to users and their individual characteristics. For example, it allows to perform a flight booking in the name of the traveler or limit usage of the application to regular employees.

2.1 Business-Process Management in SOA

Business processes orchestrate the behavior of services and humans. To this end, they define a control flow and a data flow between the entities involved, based on message passing. For example, a travel-booking BP would include an application by the traveler and approval by a manager. A flight-booking service exists indepedently of the BP, but the BP coordinates its interactions with the traveler and the manager. It performs the booking automatically, but only if the trip has been approved. It also passes it the trip details confirmed by the traveler. The WfMC reference model [17] defines an architecture for BPMS that offers this functionality. See Figure 1. The central component is the BP execution engine (*engine*). It has several interfaces. Interface 1 is between the engine and the BP modelling tool. It facilitates the deployment of process definitions. This includes their security configurations. Interface 2 connects the worklist handler to the engine. The worklist handler performs interactions

of the BPMS with users via *human tasks*. Note that there usually is a single worklist handler for all users. Users can choose from different tasks, which are often available to different users, but only performed by one user eventually. Interfaces 3 and 4 connect the engine to applications and other BPMS, respectively. These interfaces are very similar at the technical level, and the distinction mainly is for 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, process definitions commonly use WS-BPEL [36]. Applications and other processes are provided as web services using SOAP [45].

WS-BPEL only allows to express the control and data flow directly. Other aspects of the workflow, especially non-functional concerns such as service selection, access control, or logging are tedious to specify manually. In particular, specifying aspects that affect the process as a whole is difficult. E.g., to select a service once and then perform several calls to it, in standard WS-BPEL one needs to specify the service selection itself, including steps such as checking the trustworthiness of selected services and asking the user for his choice, and then store the address of the selected service in a variable and manually use the stored value for subsequent calls. There are different approaches to integrate such process-level aspects more easily into the BP definition.

[12] implements non-functional process-level requirements based on aspect-oriented programming. The actual implementation of the requirements is provided in separate components called *middleware services*. However, the authors also instrument the BPEL code of the process substantially to achieve the respective functionality. [29] proposes proxy components. It also requires instrumenting the process definition. In contrast to [12], the instrumentation does not depend on the particular security functionality to achieve. It just routes communication through the proxies. This is in line with the processing model of SOAP that allows for intermediaries, cf. Chapter 3.5 of [20].

Regarding our travel-booking use case, all this means that existing BPMS allow to express its basic functionality. However, we still need to determine how to adapt its behavior based on the identity of users.

2.2 Federated Identity Management

Federated identity management enables different organizations and software components to securely exchange identity information about users. For example, FIM allows all employees to use the travel-booking application without additional accounts. Services involved, such as the travel agency, can reliably determine the identity of the traveler. In this subsection, we review FIM concepts and corresponding technologies, focussing on the SAML framework [38] and specifications building upon it. OpenID [39] is an important alternative. However, it is conceptually similar, and SAML is better suited for the web-services world.

The basic roles in an IdM 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 whom 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., define which identity providers are trusted to correctly authenticate users.

Technically, this requires exchanging configurations such as network addresses and public keys. In what follows, we assume the existence of a federation, i. e., trust relationships and operational arrangements between service providers and identity providers. We also assume that accounts of users participating in this environment exist. With respect to our use case, this means that all employees have an account at an identity provider hosted by the company. This identity provider is trusted by the company's BPMS and external service providers involved.

2.2.1 Types of identity information

There are two kinds of identity-related information that an IdP can provide: *Identifiers* are values that identify a user in a given context. *Attributes* contain some statement about the user that may be relevant for the service provider (e. g., the type of driver's license of the user). For example, the system can use an identifier to attach an instance of the travel-booking process to a particular employee. Attributes such as the job position are used to determine whether an employee may approve trips of other employees, or to choose the booking class.

In SAML, identity providers embed identity information in assertions [30]. Assertions "carry statements about a principal that an asserting party claims to be true". Different kinds of statements carry the information mentioned above: Authentication statements assert that the user has been authenticated and contain 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.

2.2.2 Providing Identity Information to service providers

As explained above, the BPMS executing the travel-booking BP and an external travel agency need to use identity information of the persons involved, i. e., the traveler and the manager approving the trip. There are different ways how they can get this identity information from the identity provider, as follows:

BPs either interact with users directly (through the worklist handler) or with web services, which in turn can act on behalf of users. In both cases it is possible to provide identity information to the BP: When the user wants to interact with a worklist handler, he needs to authenticate. As users access web-based services at many different service providers, they do not want to manage access credentials for all of them. With SSO, the user is redirected to the IdP, which authenticates him and asserts his identity to the service provider. The other case is a user authenticating to some other application through his IdP. The application acquires tokens testifying the authentication of the user and uses them in web-service calls to other service providers.

SAML supports both ways: The *Web Browser SSO Profile* [32] allows to transfer identity information from an IdP to a web frontend of a service provider. It is based on the *Authentication Request Protocol* [30] and the HTTP Redirect, HTTP POST and HTTP Artifact bindings. The SAML Token Profile of WS-Security [34] allows using SAML assertions as tokens in WS-Security [35] headers of SOAP messages. The service provider receiving the message has to validate the evidence provided by the caller according to the method specified in the assertion (e. g., that the sender holds a specific key).

All in all, assertions as defined in the SAML standard are able to carry the identity information needed to perform travel bookings. The BPMS running the travel-booking BP can get them either through SSO or special web-service calls carrying SAML assertions.

2.2.3 Services Based on Identity Information

The travel-booking BP itself uses identity information for various purposes. For example, it needs to recognize users throughout the booking process. It also needs to make sure that no user can approve his own trip. It might use the job status to determine the booking class of the traveler. Travel booking also has many building blocks identity information is necessary or useful for. Examples are the flight-booking service of a travel agency, a calendar service itinerary data is entered into, or a service that authorizes payment based on the user's consent expressed through a more secure channel.

In more general terms, we can categorize possible services based on identity information as follow: With identifiers, one can recognize users in order to provide a stateful service or to enforce authorization constraints. Attributes allow to personalize the service. For example, an application can address data for shipping or offer services available only to a certain age group. Using identity information for access control is addressed in Section 2.4.

According to [24], an *identity web service* or *identity service* for short 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 are: (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 his decision, such as the authorization of a payment. (c) Services that can take a decision based on instructions from a user. In case of business processes, this includes a service that can declare the consent of the user to terms of service based on a policy or on his choice at 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 has invoked them, by including credentials.

The Liberty Alliance has developed ID-WSF [24], a framework for identity web services. The ID-WSF Security Mechanisms specification [22] defines the use of tokens for message authentication, including tokens that specify the invoking identity. The Discovery Service specification [26] 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. ID-WSF defines an SSO Service that lets a system obtain SAML assertions as security tokens.

ID-WSF provides a framework to develop the identity-based services necessary for our travel-booking example. However, it does not provide any dedicated support to implement the overall process. It does not address how to integrate this functionality into a BPMS.

2.3 Privacy Enhancements for FIM Systems

One can easily identify situations in the travel-booking use case where the distribution of identity information should be restricted for privacy reasons. For example, consider a service where travelers can give feedback about flights. This should only be possible for actual travelers, but they should not have to reveal their identity to the service. Moreover, it must be clear for employees which data is transmitted in any situation.

The *laws of identity* [7] address these and similar concerns. They are recommendations for FIM implementations. They are the result of intensive discussions within the IdM research community. Several laws concern privacy aspects of FIM features and respective user-control mechanisms.

One law requires the system to "only reveal information identifying a user with the user's consent" (Law 1). Another one (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 BPM, this means isolation between process 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 represent desirable privacy properties of any FIM system, including a FIM-enabled BPMS.

In FIM, identifiers valid only in a specific context are known as pseudonyms. They can be different for each service provider, but otherwise be persistent or even change for each login session. Interactions between service providers require a mapping between the different pseudonyms used at each service provider. [30] defines different kinds of `NameID`s, including pseudonymous ones. Core alternatives are globally unique names like e-mail addresses and X.509 subject names, privacy-preserving persistent identifiers, which do not have any correspondence to an actual identifier and are specific to a given service provider, and transient identifiers, i. e., random and temporary values. ID-WSF also defines an Identity Mapping Service that translates references to users into alternative formats or identifier namespaces [25]. SAML or ID-WSF do not, however, address directed identifiers for different BP instances in a BPMS.

By following these guidelines, we can make sure that the FIM-enabled BPMS envisioned provides an adequate level of privacy for the travel-booking use case and other scenarios.

2.4 Access Control

Controlling access to functionality is an important topic in any application. For example, only employees may apply for business trips, and only managers may approve these trips. Moreover, not all booking services may be used for ordering tickets.

In business processes, access control mainly is about controlling who can cause the execution of process activities. This concerns both human tasks and the processing of incoming web-service calls. In a FIM context, it is natural to use attributes to achieve this, a paradigm known as *attribute-based access control* (ABAC) [46]. ABAC is a generalization of role-based access control. One possibility to express respective access-control policies is XACML. It is a well-established de-facto standard. It is based on ABAC and integrates well with SAML, and, consequently, FIM in general. Therefore we deem it a good basis for access control. XACML includes a reference architecture and a format for decision requests and results. The SAML 2.0 Profile of XACML 2.0 [33] defines an extension of XACML authorization-decision queries, so that SAML attributes can be included in decision requests. In [19] we have developed a framework, suitable for SOA, for taking access-control decisions based on different notions of trust. Our Trust PDP also supports queries that omit the subject (i. e., the service) to *discover* services that are trustworthy according to the underlying trust policy.

XACML is not well integrated with BPM technology yet. This will be necessary to protect the travel-booking application against unauthorized access. Moreover, it is necessary to prevent an employee from approving his own trip. Multi-session separation of duty [11]

is a useful concept to this end. Finally, business processes may call out to trustworty services only. Existing technology offers no straightforward solution here.

3 Design Requirements

In Section 4 we will determine how different kinds of FIM functionality can be made available to BPs and integrated into a BPMS. The overall goal of this analysis is the development of a respective BPMS. The development should accomodate issues such as privacy and intereroperability. In the following, we formulate design requirements that will serve as guidelines for the analysis in Section 4.

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 desired privacy properties of FIM applications (Section 2.3). (2) The architecture-related requirements stem from best practices in the fields of BPM and SOA, and (3) from considerations on how to facilitate implementation and system maintenance. Further, [14] 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 BP instances*: The laws of identity propose pseudonyms for users that are different for each service provider, to prevent different organizations from combining their identity information about a user. The SAML framework provides a solution through pseudonymous `NameID`s different for each service provider. However, a BPMS, acting as a single service provider, can host different independent business processes. This means that the BPMS needs to prevent several BP instances running in the same BPMS from combining their identity information.

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

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. BPM-specific functionality should only work with the generic view on FIM. This allows to use existing FIM libraries, and to update them when necessary to follow the evolution of the underlying specifications.

D4 *Standards-based architecture:* The architecture of our BPMS and its interactions with the environment should be based on established standards. This includes the WfMC workflow reference model [17], and the reference architecture of XACML [31].

D5 *Clear assignment of functionality to components:* We want to assign responsibility for a security-relevant function to one component, as done by the specifications mentioned

in D4 as well. This reduces complexity and makes it easier to guarantee correct behavior of the system.

D6 *Declarative configuration where feasible:* The BPMS offers security functionality for applications with different security requirements. Consequently, it must be possible to configure the BPMS so that it fulfills the security requirements of a specific application. We see two alternatives how to accomplish this: (1) The BPMS provides operations that an application BP 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 BPs. When performing security functionality, the BPMS evaluates this configuration, taking the state of BP instances into account. – We prefer (2), because this approach allows for better separation of concerns, i. e., between the application functionality and FIM. In particular, this eases re-use of existing application BPs, because developers do not need to touch the BP definition itself when configuring the security. This is in line with policy-driven security pursued by the WS-SecurityPolicy specification [37]. However, we expect that in some cases it is not possible to provide generic policy-based solutions because 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. We have the architecture-related ones under control insofar as they concern our BPMS and its components. Therefore, the BPMS envisioned must fully implement them. Fulfillment of the privacy-related requirements does not only depend on our BPMS, but also on the environment it is running in, and on the BP application it executes. We therefore require that the resulting BPMS does its part to fulfill them. We have already stated this assumption in general terms in D2. We will address the privacy issues of the various FIM concepts in Section 4. The design decisions in Section 5 will take these issues into account. We will assess the resulting BPMS with respect to the privacy-related design requirements in Subsection 7.1 and state in more detail the assumptions on the environment so that the system as a whole can fulfill them.

4 Requirements Concerning Support for Individual FIM Concepts

Taking the design requirements from Section 3 into account, we now examine how to integrate different FIM concepts into a BPMS. The discussion addresses the following aspects:

- *Data flow:* When implementing the FIM concept, on which occasions does identity information enter or leave the BPMS? Which components of the FIM infrastructure are involved? This affects the interfaces of the BPMS.
- *Context used:* Which types of BP context, such as the execution history of a BP 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 login session of the user, the lifetime of a process instance, or a single activity. The answer affects where such information should be stored.

- *Need for configuration:* Does the functionality work for BPs without explicit security configuration, or is it necessary to include a specific security configuration in each BP definition?
- *Type of configuration:* Is entirely declarative configuration possible (cf. D6), or do BP 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 BP instances.
- *Basis for configuration:* Which elements of the process definitions must the configuration refer to, and what is the runtime information corresponding to such references? For example, a separation-of-duty constraint refers to activities in the BP 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 BPMS handle privacy issues (D1 and D2) arising with respect to the functionality in question?

By analyzing each FIM feature with respect to these aspects, we have been able to develop the architecture presented in Section 5 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 7.

The following subsections discuss the FIM concepts presented above in the context of BPs. Subsection 2.2.2 has covered different ways how identity information can reach a relying party, i.e., the BPMS in our case. They are covered in Subsection 4.1 (SSO) and 4.2 (incoming WS calls). Subsection 2.2.3 has addressed different ways of using this information, covered one by one in Subsections 4.3 through 4.5. Finally, 2.4 has addressed the combination of FIM and access control. Subsections 4.6 and 4.7 address this in relation to individual process activities and the entire process, respectively.

4.1 SSO

This subsection discusses how to perform SSO for the user interfaces of business processes.

Data flow: In a BPMS, users interact with the worklist handler to perform human tasks through a web-based interface. Identity information enters the BPMS when the user logs into the tasklist handler using SSO. *Context used:* At this time, information about BP instances is not yet relevant. The information acquired through SSO is merely stored for later use. *Lifetime:* The information acquired through SSO is needed at least for the duration of the respective SSO session. During this session, the user may view his task list and perform one or more tasks. When the user has performed a task, the BPMS creates a relation between the corresponding BP activity and the identity information of the user. The required lifetime then depends on the purpose the information is used for, as we will assess below. *Configuration:* How the BPMS uses the information acquired depends on the tasks performed and on the corresponding BP definitions. We do not see any need, however, to configure how to perform SSO itself. *Privacy issues:* SSO causes the identity provider to pass identity information to the BPMS. This has obvious privacy implications. Therefore, the user must be able to trust the BPMS, and he must be aware of any disclosure of his identity information. As long as the user does not perform any tasks, the BPMS does not need to provide any identity information to BP instances. Of course, the user has to trust

the BPMS itself not to disclose the information and to protect it against attacks. The BPMS must make clear which BP instances identity information is potentially provided to when the user performs a task. Further, the BPMS has to provide access to the privacy policy of that BP. Moreover, the BPMS could perform automatic checks based on a consent policy submitted by the user.

4.2 Incoming Identity-WS Calls

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

Data flow: In the infrastructure envisioned, web-service calls can carry identity information, pointing to the individual in whose name the caller invokes a service. BPs 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. *Context used:* Each web-service call invokes a specific activity in a specific BP instance. The BPMS 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 initialized by the first message of the conversation. Subsequent incoming messages are then routed to the same BP instance. The context needed here is the content of the correlation sets of the running BP instances. *Lifetime:* The identity information included in the incoming call relates only 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 authorization on the activity level, it is not needed afterwards. In any case, identity information is no longer needed once execution of the BP instance has finished. *Configuration:* Correlation sets, which determine the BP 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. *Privacy issues:* When a call is received, BP instances directly get identity information. It would not be convenient for the user if he had to access the BPMS and give consent. This means that the caller has to make sure the user has consented before making the call.

4.3 Using Attributes for a Personalized Service

Business processes 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 attribute values of the user can help to customize the service to him. For example, a BP for booking a rental car can exclude premium-category vehicles when the driver's license of the user is less than two years old. We now discuss how to facilitate such customizations.

Data flow: The BPMS needs to provide identity information relating to BP activities (Sections 4.1 and 4.2) to the corresponding BP instances. *Context used:* Identity information always belongs to a single user performing tasks in a process instance. A BP instance can acquire different attributes for the same user with respect to different BP 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 as well, not only the user whom 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. *Lifetime:* A BP instance potentially needs identity information acquired by a specific activity at any point in its lifetime. *Configuration:* The actual customizations 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. The BPMS does not know how to process the attributes. This completely depends on the BP definition. It is best specified using imperative process logic, because there are no generic patterns that can be a basis for declarative configuration. The task of the BPMS solely is to provide user attributes to BP instances, which process this information according to the process definition. We will assess different ways of providing attributes in Section 5. Configuring declaratively *which* piece of information to retrieve appears feasible. However, the resulting data needs to be inserted into the data flow of the process, which is connected to the imperative application logic. To this end, we see a need for explicit activities that retrieve identity information and make it available to the application logic of the process. This can be accomplished by letting them refer to another activity in the process, and thus the identity which has performed that activity. *Privacy issues:* A user must be aware which attributes are provided to a BP instance before he triggers the actual transfer. Given this, it is implied that the instance can use the attributes.

4.4 Recognizing 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 BP. 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.

Data flow: When a user accesses the BPMS a second time, it has to recognize him based on the identity information provided. *Context used:* The BPMS must know the identifiers of users who have performed activities in the past, so that it can compare them to the one of the user performing the current activity. *Lifetime:* The interaction of an application with a user usually covers the entire lifetime of a process instance, so the BPMS needs to remember identifiers of past activities until the instance has completed. *Need for configuration:* Applications have tasks that the same user should perform, so one must be able to specify which tasks are concerned. This is a kind of process-level authorization constraint; we consider it below in Section 4.7. *Privacy issues:* With this functionality, application BPs can learn that a user is the one who has performed a previous activity. However, this is necessary to maintain a stateful interaction, so that users can reasonably expect such a use when they perform an activity and provide identity information to a BP.

4.5 Invoking Services on Behalf of Users

In a framework based on identity web services, BPs invoke such services on behalf of a user. This enables them, say, to access personal data in a personal data store of the user. To this end, it is also necessary to find services available to that user and to 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.

Data flow: When invoking services on behalf of users, the BPMS 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 makes the call on behalf of this user when it actually performs the call. *Context used:* The user on whose behalf the BPMS makes the call depends on the activity. He must have performed an activity in that process instance before, so that his credentials are available to it. The BPMS 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 BPMS is the activity performing the call, and a previously performed activity in the same BP instance together with its identity information. *Lifetime:* The BPMS has to remember credentials of users who have performed activities in the process instance. *Configuration:* A characteristic of BPs 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. E. g., approval by another user might be necessary. Accordingly, it is not immediately clear on *whose* behalf the BPMS needs to make a call. For each outgoing call to an identity web service in BP definitions, it is necessary to configure which identity information to use. This identity information comes from a previous activity performed by the respective user. It is possible to refer to a previous activity by specifying its name. 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 leave respective solutions as future work. *Privacy issues:* When invoking services on behalf of users, this issue is twofold: First, the BP transfers identity information to the services invoked. The BPMS alleviates corresponding 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 policy of the user. Second, the services invoked may expose personal data of the user. In principle, the user consents to this by using the BP, but appropriate control mechanisms are needed. However, note that this applies to any chaining of service calls and is not BP-specific. In addition, the invoked service may check whether it trusts the BPMS to convey user consent correctly.

4.6 Activity-level Authorization

This subsection examines how to perform authorization, taking only the current activity into account. ABAC uses attributes of users to decide whether access on some entity should be granted.

Data flow: In BPM, ABAC concerns human tasks as well as service calls directed to process instances in the name of a user. *Context used:* The BPMS has to use the attributes associated with the incoming call or provided via SSO to the worklist handler. *Lifetime:* The access-control decision can be taken for each such process activity individually. This means that identity information is used immediately for access control and is no longer necessary afterwards. *Configuration:* The attributes required depend on the application, so each BP definition needs an access-control configuration. A declarative specification, e. g., an XACML policy, is fit for this purpose. It specifies the attribute values a user must have to perform each activity.

4.7 Process-level Authorization

BPs are stateful and can involve several users performing tasks. Authorization needs to take the relationship between tasks into account. This subsection therefore discusses how to support such authorization methods in our architecture. The fundamental concepts that cover

most application needs are separation of duty (SoD) [21] and binding of duty (BoD) [43]. SoD requires different users for conflicting activities, such as *Authorize payment* and *Issue cheque*, while BoD requires the same user to perform several activities. We always apply these constraints on complete instances of business processes, i.e., the most refined business context, using the terminology of [11]. More complex relationships are possible, e. g., based on attribute values. Their implementation is possible following the same basic approach.

Data flow: As in the previous subsection, process-level authorization concerns human tasks as well as service calls directed to process instances. Authorization decisions use identity information that have entered the BPMS for different activities, either through SSO or incoming web-service calls. *Context used:* To facilitate SoD or BoD, the BPMS needs to know who has performed past activities in the same BP instance. *Lifetime:* The information is potentially needed through the lifetime of the BP instance. *Configuration:* The SoD or BoD constraints that exist between the activities of a BP are application-specific. Accordingly, BP developers must be able to specify the constraints for each BP definition, i. e., the activities that must be performed by different users (SoD) or the same user (BoD). 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 leave respective solutions as future work. *Privacy issues:* The BPMS can only enforce SoD and BoD constraints correctly with persistent pseudonyms. They allow the BPMS to recognize users permanently.

4.8 Summary

We have systematically explored the implications arising from the integration of FIM functionality into a BPMS. In particular, we have disclosed how the BPMS processes identity information, and which context and configuration options are needed. This paves the way for the design of an extended BPMS in the next section.

5 System Design

In this section, we derive the structure of an extended BPMS for identity BPs, based on the requirements from the previous section. First, we examine which functionality a generic FIM layer can perform, in line with D3, and how BPMS components can use this layer. In line with D5, we then introduce new components of an extended BPMS responsible for acquiring identity information, storing it, and performing functionality based on it, such as access control. We define the interactions between the components. In particular, we define how BP context reaches the components that need it. We also describe the configurations necessary for BPs running in the extended BPMS.

5.1 Basic Architecture

Figure 1 shows a simplified version of the BPMS architecture in SOA and its interfaces according to the WfMC reference model [17]. One has to extend it in several ways: Interfaces 2–4 concern the communication of BPs with the outside world at runtime. BPs send and receive WS 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 WS interface so that identity information is acquired. The core BPMS has to process that information to provide the FIM

functionality described in Section 4. Interface 1 concerns the deployment of BP definitions. Security configurations must accompany them, so one has to define the structure of this configuration and how it is distributed to the components that need it.

Based on the design decisions explained in the following subsections, we will extend this architecture. Figure 2 shows the extended architecture. New BPMS components have a grey backgrounds. The diagram is structured into several parts: The main part of the BPMS, the engine that executes BP instances, is at the top. Below are the other components, including our new security components. The left part contains component specific to the handling of human tasks. The right part in turn deals with web services. The space in between contains components that are common to both groups. Components that are not part of the BPMS but of the surrounding infrastructure have a dashed edge. They are shown at the bottom of the diagram.

5.2 Encapsulation of Technology-specific FIM Functionality

Problem: To conform with D3, technology-dependent FIM functionality should be encapsulated in a separate layer. The BPMS envisioned relies on different kinds of FIM functionality. We can partition the functionality required 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 BPMS acquires it either through SSO in case of web-based interactions, i. e., a user performing human tasks through the worklist handler, or when it receives identity-WS calls through its web-service interface (Sections 4.1 and 4.2). (2) Outputting identity information: The BPMS has to use identity information when invoking web services on behalf of users according to ID-WSF, and when requesting activity-level authorization decisions using [33] (Sections 4.5 and 4.6). The exact behavior for all this depends on the technical specifications mentioned. (3) Accessing parts of identity information: For some tasks, the BPMS itself needs to access parts of identity information. In particular, it needs to access attributes to provide a personalized service, and identifiers to recognize users in a stateful interaction and to perform process-level authorization (Sections 4.3, 4.4, and 4.7). Consequently, we need a FIM layer implementation that can acquire and output identity information while encapsulating the implementation details, and allows access to attributes and identifiers. Because several components of the BPMS 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: Several libraries that provide the desired functionality are available. ZXID [3] is a library that implements the ID-WSF protocols. Other possibilities are the Liberty Open Source Toolkit[6], another implementation of ID-WSF, and OpenSAML [2], which implements only SAML itself, not the ID-WSF that is built on top of it. Another alternative would be to implement a suitable library ourselves.

Discussion: 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 allows to address it 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 deleted explicitly. ZXID can make outgoing ID-WSF calls and request authorization 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.

Conclusion: ZXID is a library under active development and continuously integrates new features. It also supports ID-WSF. Thus, we deem it the library best-suited for our needs. Because ZXID stores sessions in the file system, components accessing the FIM layer must run on the same machine. However, we deem this acceptable for the start. The alternative, implementing a library supporting distributed setups from scratch, is impractical because of the amount of work it would require. Our choice therefore is ZXID. An option for later development is to add support for passing the content of ZXID sessions between different installations.

Impact on the Architecture: ZXID is used throughout the architecture. First, several component use ZXID for external communication that uses a special protocol supported by ZXID. This is indicated in Figure 2 by a small circle at the respective end of an arrow. Second, those components access identity information (identifiers or attributes) already stored in a ZXID session. Components are marked by a small star to indicate this.

5.3 External Communication and Acquisition of Identity Information

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

Alternatives: The first option is to use proxy components. All external communication of BPs 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, one can instrument BP definitions so that BPs themselves perform this functionality. We have presented these different approaches to integrate non-functional concerns into BPs in Section 2.1.

Discussion: Requiring BP instances to handle the respective protocols would require complex and technology-specific instrumentations. In 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 5.2.

Conclusion: We decide to have two dedicated proxies to isolate the BP 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.

5.4 Access Control

In Subsection 5.3, we have already introduced components that intercept all communication between the BPMS and its environment. These proxies can selectively block messages for security reasons. In this subsection, we now have to take several design decision concerning the access-control part of our architecture. This concerns the partitioning of the system as a whole into components, namely for policy decisions and policy enforcement, for human tasks and web services, and for activity- and process-level authorization constraints.

5.4.1 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 them 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 decide to assign access-control *enforcement* and policy-based access-control *decisions* to separate components.

5.4.2 Decisions for Different Kinds of Activities

Problem: The question is whether to use separate PDPs for different kinds of activities.

Alternatives: Alternatives are a separate PDP for human tasks as well as for incoming WS calls and the respective PEPs, or one combined PDP.

Discussion: For both kinds of activities, access-control decisions are based on the attributes and on the identifiers of the users who try to perform them. Moreover, process-level authorization constraints are not limited to one kind of activities.

Conclusion: We use one combined PDP for all activities.

5.4.3 Structure of Policies

In Subsection 4.6 and 4.7 we have addressed the need to authorize the execution of activities both on the activity level and on the process level. The BPMS should perform both kinds of authorization based on policies. Depending on the structure of the policies, we see several options concerning the components responsible for the evaluation of the policies.

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

Alternatives: One option is to separate the policy in two parts. The alternative is to allow mixing two kinds of constraints. With both options, one can express the same constraints, but the latter option allows to group constraints of both kinds. Application developers may find this more intuitive. Process-level constraints are expressed as predicates over tuples of activities. We have to decide what kind of constraints we allow.

Discussion: The respective parts of the security specification are usually independent, see Chapter 8 of [5]. Having two independent policy parts allows their independent evolution. In particular, this is important because languages for stateless activity-level policies are widely established and not BP-specific, whereas languages for process-level constraints are not yet standardized to the same extent. Most approaches for process-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 XACML policies [11]. However, this requires specifying each constraint in the policy multiple times for each activity concerned. Thereby, 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 process-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.

5.4.4 Structure of the Policy-decision Point

Problem: Having decided to separate the activity- and process-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. In contrast, process-level constraints take the execution history into account. On the one hand, a monolithic PDP can have better performance, because it only needs to access identity information once and saves communication overhead. On the other hand, a modular PDP is more flexible: First, one can change the policy language. Second, the stateless part of the policies heavily depends on the concrete technology. 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 regarding D3.

Conclusion: We choose 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 BP-specific process-level constraints, called *Policy Decision Point for Business Processes* (PDP-BP). See the middle part of the architecure diagram. (2) For the stateless part, any existing PDP compliant to the SAML profile for XACML [33] can be used. – The PDP-BP library 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-BP.

5.5 Connecting Identity Information with Activities

Identity information, whether acquired via SSO or incoming WS calls, is connected to a specific execution of an activity in a specific BP instance. The BPMS uses it immediately for authorization of this activity (Sections 4.6 and 4.7), but also needs it later on, e. g., to invoke services on behalf of the user or to perform process-level authorization 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.

5.5.1 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 WS calls.

Discussion: On the one hand, establishing the relationship between identity information and an activity works quite differently for human tasks and WS 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, process-level authorization constraints can involve activities of both kinds. It is easier to evaluate them when the necessary information is stored in one place.

Conclusion: We decide to use one component for storage, because it is necessary to combine the information for human tasks and web-service calls anyway to evaluate process-level constraints.

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

5.5.2 Human Tasks

In a traditional BPMS, the execution of human tasks works as follows: An activitity in a BP 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 BP instance. Our envisioned BPMS 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 5.1 (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 Authorization"), the payload for the human task (i.e., the name of the traveler, 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/AuthorizeTravel.Callback/`*, the name of the activity ("Authorize Travel"), and the ID of the BP instance (e.g., "BP2342").*

Problem: Establishing the relationship between identity information and activities has two facets: The first one 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 one concerns establishing this relationship before it is stored in the PIP-BP. These facets are independent because 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 BPMS, except for the PEP-HT. This means that it sends all access requests to the PEP-HT, which in turn forwards them to the PDP-BP. The PEP-HT also registers completed human tasks in the PIP-BP. 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-BP and the PIP-BP requires changes that are less specific to our architecture. In any case, we have to extend the worklist handler to support SSO and to request an access-control decision before allowing users to see or perform tasks. Were the worklist handler to call the PDP-BP directly, it would have to know which activity in which process instance had created a task, and support the protocol for such requests.

Conclusion: We decide to extend the worklist handler only minimally, and assigning additional functionality to the PEP-HT. This means that the PEP-HT forwards access-control requests and registers completed tasks.

Impact on the Architecture: The worklist handler is extended with SSO functionality by using ZXID. A circle in the diagram indicates this. For each human-task-instance, the PEP-HT creates an ID, stores it and the endpoint reference of the callback in the PIP-BP, and forwards the request to the worklist handler. Users can login into the worklist handler

once using SSO and then perform several tasks. Here, authorization has to occur twice: When a user views his worklist, the worklist handler must know whether to include a given task. When a user actually performs the task, authorization is required again because of constraints. Think of a BoD constraint between two tasks. As soon as a user has performed one of them, other users are no longer allowed to perform the other task. Whenever a task is completed, the worklist handler includes the correspoding ZXID session ID in the response. The PEP-HT stores it in the PIP-BP before forwarding it to the BP instance. The architecture diagram shows an arrow between the PEP-HT and the worklist handler. It stands for creation and completion of human tasks and for authorization requests. the PEP-HT is also connected to the PDP-BP (for authorization requests) and the PIP-BP (to store which identity information belongs to completed tasks).

5.5.3 Web Services

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

Example 5.2 (Incoming WS 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 process notifies this application about new requests for business trips using a web-service call. Managers view the list of open requests at some later time, and approve trips based on the overview provided by the application. The application sends the results back to the BP 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 BP, so that business trips would be booked without approval.*

Problem: As explained in Section 4.2, BPEL provides correlation sets to determine the context of the call, i. e., the activity and BP instance the call is directed to. Normally, it is the BP engine that performs correlation. However, the BPMS needs this context to perform authorization, and to establish the relation between identity information and an activity, to store it in the PIP-BP.

Alternatives: We see several approaches to address this problem: (1) Re-implementing correlation in the PEP-WS. (2) Instrumenting the process, activities receiving calls in particular: The idea is to insert a process fragment that sends the ID of the BP instance and the name of the receiving activity to the PEP-WS. With this information, the PEP-WS can perform authorization and send the authorization result as a reply. If authorization is denied, the received message is discarded, and the receiving activity is started again. (3) Close integration of the PEP-WS with the BP engine to perform correlation without actually delivering the message. (4) A minimal solution supporting only calls that start new BP 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 process instrumentation.

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

Conclusion: Our pragmatic solution is to start with (4) and leave (2) as future work. Appendix A features a detailed illustration of the process instrumentation necessary for (2).

Impact on the Architecture: The PEP-WS receives WS calls using ZXID, indicated by a small circle in the diagrams. The PEP-WS then sends a request to the PDP-BP. The request contains the activity and BP 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 BP 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-BP.

5.6 Outgoing Web-Service Calls

For outgoing web-service calls made on behalf of users (Section 4.5), the BPMS first has to determine which service to call. Then it has to make the call. To this end, it has to include the correct credentials. Because we rely on unidirectional identifiers valid for one service provider only, the BPMS 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, it 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 [23]. 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.

5.6.1 Determining the Identity

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

Alternatives: We see two main alternatives to solve this problem: (1) When activities are performed, the BP instance gets a token representing the identity information acquired. (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 WS call. The BPMS then uses identity information from a_r to perform a_i.

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

Conclusion: Given the arguments just provided, we choose Alternative (2).

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

Example 5.3 (Web-service call using identity information) *In our earlier example, a_r is a human task where 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 he agrees to the use of his credentials to perform the call to the booking service.*

5.6.2 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 personalized trust policies.

Alternatives: We see two alternatives to deal with these policies. (1) The component that stores the policies (policy store, PS) and Trust PDP are separate components. (2) The Trust PDP knows the trust policies of users. Note that, in a distributed setting, users should be able to choose a Trust PDP. Thus, it depends on the user which Trust PDP is used.

Discussion: (1) discloses the trust policy to the BPMS, (2) does not. In addition, (2) is more interoperable, because there currently is no common trust-policy language in widespread use.

Conclusion: Our current implementation is similar to (1). The reason is that we use an existing Trust PDP that supports dynamic policies passed at runtime, but does not store user policies itself. At the moment, we only use one central Trust PDP. This means that users cannot choose among several ones. An advantage is that a central Trust PDP has access to more user feedback and can thus deliver more accurate trust rankings.

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 1.

5.7 Providing Attributes to BP Instances

Problem: BPs must be able to access attributes of users to provide a personalized service (Section 4.3).

Alternatives: There are different ways to accomplish this [15]. They include (1) extending the interfaces of the BPMS so that all attributes available are provided once the BP instance receives a call, or a human task is completed, or (2) letting BP 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 BP does not need them. There is no need for BP instances accessing 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-BP.

Conclusion: We choose to implement Alternative (2). This option is very flexible and allows BPs to access identity information regardless of how it has entered the BPMS.

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

Example 5.4 *In our travel-booking example, we assume that managers are allowed to travel business class, while other employees are limited to economy class. Thus, the BP has to determine the job role and pass the corresponding booking class to the flight-booking service. We assume that the identity provider passes the employment status as an attribute. To this end, the business process sends a request to the IdI Access service which includes the BP-instance ID, the name of the activity ("Initiate booking"), and the name of the attribute ("*`job_role`*"). The IdI Access service replies with the value of the attribute as it has been acquired by the specified activity. Depending on the value, which is either "manager" or*

"regular_employee", the BP chooses the booking class and includes it in the call to the flight-booking service.

5.8 Configuration

In the previous subsections, we have insinuated that the components of the system need to be configured to perform the functionality required. We now summarize which configurations are needed and determine how to make them available to those components of the BPMS that need them.

In Section 4, we have examined the necessary configurations for FIM functionality used in a BPMS. Earlier in Section 5, we have determined the components responsible. This has resulted in the following types of configuration used by our BPMS envisioned:

Activity policy: Access-control policies for activities, consisting of two parts, namely for the authorization on the level of individual activities (*activity-level constraints*) and constraints involving several activities (*process-level constraints*). The separation into two parts suggests itself given that the two parts are independent and evaluation works quite differently. It is similar to the approach in [4]. – Note that trust policies of users are not part of the configuration of the BPMS. This is because the BPMS just retrieves them from a policy store and forwards them to the Trust PDP. An obvious way to express activity-level constraints are XACML policies. These policies need to specify which subjects are allowed to perform which actions on which resources. XACML uses attributes to describe all these. More explicitly, the resource is specified by using activity names as a resource attribute. Unlike in [4], the policy is not limited to roles for describing subjects. Instead, an arbitrary combination of attributes can be used, which allows more flexible policies. Solutions for process-level constraints include BPCL from [4], or a simple set of BoD and SoD constraints each referring to two or more activities.

Example 5.5 (Two-part access-control policy for activities) *The first part, i. e., activity-level constraints, can look as follows. Because XACML is very verbose, we state this part in natural language. The 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 = "Authorize Travel"` *and action "perform", require subject attribute* `job_role = "manager"`*.*

The second part, i. e., process-level constraints, is a set of BoD and SoD constraints:

- *BoD("Initiate Booking", "Choose Airline"): The employee initiating the travel-booking process must confirm the details of his own trip.*
- *SoD("Initiate Booking", "Authorize Travel"): No employee, not even a manager, may approve his own trip.*

Identity-selection policy: For each activity performing an outgoing WS call, the identity information to use has to be given by specifying the activity that has acquired it. This is just a set of pairs of activity names, such as $\{(\text{Confirm Booking}, \text{Book Flight})\}$. This 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.

We now describe how the configurations just described can be deployed to the BPMS. When a BP definition is deployed to the BP Engine, the identity-selection policy has to be deployed to the PDP-BP, using the identifier of the BP definition in the BP Engine. There are alternative ways to provide the activity policy to the stateless PDP: (1) It is deployed to the PDP-BP, which provides it with every authorization request. This is not possible with every PDP implementation and can be inefficient. (2) The policy part relating to individual activities is deployed directly to the stateless PDP. Requests made by the PDP-BP to the stateless PDP include the name of the BP definition so that the stateless PDP can determine the policy applicable. – Our current approach is based on (1), but we plan to switch to (2) for efficiency reasons.

6 Implementation

As mentioned above, we rely on the ZXID library [3] for functionality. ZXID is written in C but provides bindings for several languages, including a Java binding based on the Java Native Interface (JNI). Its functions for performing SSO or processing incoming WS calls can be integrated into J2EE servlets as well as other technologies for serving HTTP requests. We use our own implementation of the worklist handler, because existing implementations are either closed source, not mature enough or too complex. The human-task definitions of our implementation are basically form descriptions written in XML, where form elements can be marked as input, output, or both. Our worklist-handler implementation automatically generates WSDL interface descriptions for each human task.

The PEP-WS and PEP-HT are implemented as J2EE servlets, making it easy to integrate ZXID. As the BP Engine, we use Apache Ode. We currently have to instrument BP definitions manually before deployment, replacing calls to external web services and the worklist handler with calls to the PEP-WS and PEP-HT, respectively. Our PDP-BP uses the PERMIS PDP [1] as its underlying stateless PDP. The PERMIS PDP allows passing the policy at run-time, a feature intended to be used with so-called sticky policies. The interfaces the PEP-HT, PIP-BP and PDP-BP provide to other components are implemented as SOAP web services. The PIP-BP uses a MySQL database for storage.

7 Evaluation

In order to evaluate the architecture created, we pursue a twofold approach. We first assess it with respect to the design requirements presented in Section 3. We then demonstrate in a case study that it is feasible to model a business process that uses the functionality.

7.1 Assessment

In Section 3, we have introduced a number of design requirements. We can group them into privacy-related (D1 and D2) and architecture-related ones (D3 through D6). We will now assess our architecture with respect to these design requirements by means of plausibility arguments.

D1 requires preventing BP instances from combining identity information. We allow BP instances to access *any* identity attribute of users which could serve as quasi-identifiers. Therefore, BP definitions have to be validated before deployment to ensure that they adhere

to their privacy policy. In particular, BPs may only correlate identity information from different sources when necessary to achieve the purpose of the BP and specified in that policy.

According to D2, identity information should be released only with the user's consent. Regarding 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 BP, 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 BPMS. We see this component (the caller or the identity provider, respectively) responsible for ensuring that the user has consented to this transfer. The BPMS itself is responsible when it provides identity information to BP instances or performs calls to external web services that include identity information. Identity information included in web-service calls is directed to a specific BP instance. The BPMS provides it to this BP instance without further checks. Identity information acquired through SSO is only provided to a BP instance when the user performs a task in that BP instance. Our BPMS does not yet provide fine-grained control of the attributes that the BP instance can access. For example, the travel-booking BP can access the `drivers_license` attribute even when the trip does not include usage of a rental car. A respective solution should leverage application semantics to be practical. We sketch the requirements on such a solution below. When the BPMS invokes a web service, it checks that the service is trustworthy according to the user's policy (Subsection 5.6.2). The identity provider issues an assertion containing the user's identity information specifically for the service invoked. This means that it can control which information to release. – In summary, we see two major opportunities for improvement: (1) Identity providers should have a mechanism to control which identity information they disclose. (2) The BPMS should allow user to control which attributes a BP instance will be able to access. A possible way to achieve this is annotating BP definitions with the set of attributes they potentially access, computing the intersection with the user's attributes, and displaying a message when a user is about to perform a task.

D3 postulates abstration 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 concrete specification such as SAML. Secondly, the architecture uses a dedicated software layer to encapsulate the details of the technology actually used Subsection 5.2). 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 BPM functionality, and the XACML reference architecture for its access-control functionality. Following existing standard architectures is also one important step towards the a clear assignment of functionality to components. Following the WfMC reference model makes sure that the functionality needed is implemented for all interfaces of the BPMS. In line with the XACML reference architecture, the tasks of policy enforcement, policy decision, and storing information needed to this end are assigned to dedicated components (Subsections 5.4.1 and 5.5.1). Our architecture specification also defines whether common or distinct components are

responsible for similar functionality regarding human tasks and web services, and specifies how they interact (Subsections 5.5.2 and 5.5.3).

According to D6, declarative means of configuration should be preferred over explicit control of FIM functionality by individual BPs. The architecture employs declarative policies for access control and determining the identity to use in web-service calls (Subsection 5.8). These policies are independent from the BP definition itself, so it is easy to adapt them when more powerful specification mechanisms become available. When BPs need attribute values for application-specific purposes, they have to fetch them explicitly. We conclude that the BPMS uses declarative policies wherever this is possible in an application-dependent way.

In summary, the BPMS fulfills the architecture-related design requirements. Regarding the privacy-related requirements, there is room for improvement, but this requires the development of a mechanism that allows to specify the identity attributes needed by an application, or the integration of an existing one. This is mainly implementation work. Because it has an impact not only on our BPMS, but also on BP-based applications, a conceptually sound specification needs to be developed.

7.2 Case Study

In the following, we provide a case study based on the travel-booking example. We demonstrate how the corresponding security specification is evaluated during BP execution.

The example BP definition, dubbed P, is a linear sequence of four activities. a_1 (*Initiate booking*) receives incoming WS calls that starts the BP. The interface comprises a destination, travel dates, and a reason as input. a_2 (*Authorize travel*) is a human task, which displays destination, travel dates, reason, and the name of the traveller, and asks for authorization. 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 BP definition is accompanied by the following security configuration: (1) Activity level authorization: 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) Process-level authorization 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 BPMS executes a_4 in the name of the user who performed a_3.

We now list the steps in an exemplary execution of P together with the component interactions they cause:

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

- The BP Engine starts 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 it in the PIP-BP, 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. ZXID handles the SSO process and stores Bob's IdI. For all available tasks, including h_1, the worklist handler requests an authorization decision from the PEP-HT. The PEP-HT determines that h_1 had been created by a_2 in P_1 by asking the PIP-BP. Then the PEP-HT sends an authorization request, which contains Bob's IdI, a_2, and P_1, to the PDP-BP. The PDP-BP's policy states that a SoD conflict with a_1 exists. It looks up the IdI for a_1 in the PIP-BP and determines that a_1 was performed by a different identity, namely John. It then requests authorization from the stateless PDP. This is granted because Bob is a manager. The worklist handler gets the PDP-BP's decision via the PEP-HT and shows h_1 in Bob's worklist. Bob chooses h_1 and completes it, authorizing John's trip. Because new authorization constraints might apply due to activities executed in parallel, the worklist handler requests authorization again, which 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 (i. e., 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 BP Engine.

- The BP Engine starts 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-BP, 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. ZXID handles the SSO process and stores John's IdI. For all available tasks, including h_2, the worklist handler requests an authorization decision from the PEP-HT. The PEP-HT determines that h_2 had been created by a_3 in P_1 by asking the PIP-BP. It then sends an authorization request, which contains John's IdI, a_3, and P_1, to the PDP-BP. The PDP-BP's policy states that a BoD relation with a_1 exists. It looks up the IdI for a_1 in the PIP-BP and determines that a_1 was performed by the same identity.It then requests authorization from the stateless PDP. This is granted just like for a_1. The worklist handler gets the PDP-BP's decision via the PEP-HT and shows h_2 in John's worklist. John chooses h_2 and completes it, choosing Universum Airways. The worklist handler requests authorization 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 sents 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 BP Engine.

- The BP 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-BP which identity to use for the call. The policy states that the identity from a_3 shall be used, so the PDP-BP looks up who has performed a_3 in P_1 in the PIP-BP. It sends the result, a pointer to John's IdI, to the PEP-WS. The PEP-WS now performs service discovery using ZXID and discovers two flight-booking services, b_1 and b_2. It then retrieves John's trust policy from his trust-policy store. It 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. ZXID performs identity mapping automatically.

One can see that a simple BP together with a simple policy for it lead to many actions performed by the BPMS in the background. This would be very tedious for BP designers to specify explicitly. Our BPMS in turn performs the required actions automatically. This facilitates using FIM for business processes and improves reliability with respect to identity management.

7.3 Discussion of possible extensions

Regarding some concepts, the architecture described in this article features only basic solutions. We acknowlege that there are more sophisticated solutions available for specific aspects in isolation and described in the literature. In the following, we sketch approaches to integrate them with our architecture. However, a full integration is outside the scope of this article.

7.3.1 Delegation of Authority

One of our requirements is that the business process must be able to invoke services on behalf of users (Section 4.5). This requirement, also known as delegation of authority, has been studied in depth in the literature. According to [9], an important requirement on the conceptual level is the fine-grained decision of which privileges, attributes, or roles are to be delegated. In addition, delegation should be subject to an express delegation policy. The authors propose 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.

In contrast, we rely on the service discovery specified in ID-WSF. When a user logs into the BP engine using SSO, the IdP supplies a token that allows using the discovery service and the identity mapping services. When a BP needs to invoke a service with privileges of the user, it possibly has to use the discovery service in order to find a concrete 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 delegation, however, is not controlled by a separate component, and there are no explicit policy checks.

In order to use the above-mentioned delegation service with our architecture and determine which privileges users need to delegate, BP designers would have to specify the following: Which kinds of services does the BP need to invoke on behalf of users? Because concrete service endpoints might only be known after service discovery, they cannot be part of the specification. Which data does the BP 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 BP 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 BP 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 BP designers specify the required privileges separately for different branches of the BP. However, this would require repeatedly asking the user for delegation, and the BP would be blocked until the user actually performs it. From a user interaction perspective, we argue that the delegation service must be able to handle delegation requests by the BP, which the user can then confirm using, say, a web interface.

7.3.2 Generic security architecture

We have described an architecture based on policy-enforcement points for the interfaces of a BPMS and a dedicated PDP-BP which can handle the state information to be used in access-control policies for BPs. There exist generic proposals for security enforcement architectures. One such architecture is [10]. 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 article in contrast deals with the special security requirements of BPs 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 BP could be combined with sticky policies for the different user data accessed in the BP. However, this would exceed the scope of this article. Mapping the concepts developed here to a more generic architecture is therefore left as future work.

8 Related Work

[15] proposes extensions for a BPEL engine to let processes access the content of security tokens. This approach only considers web-service interactions and requires BP definitions containing explicit activities that access the information. Our approach also addresses user interactions (through human tasks). It covers the whole range of FIM concepts and allows declarative configuration of respective mechanisms. In fact, [15] is one possibility to give processes access to profile information contained in tokens (see Section 5.7).

[18] presents an approach for identity and access management in SOA. Their policy language can express RBAC models with ressources, operations, permissions and entities called *context*. It also supports SoD, but only in the static form. Scopes in BPEL processes are annotated with the role required and the context valid in that scope. Similarly to our approach, the annotated BPEL processes are automatically transformed to enforce the policies. To this end, the process retrieves user name and password of an eligible user from a *credential service* and acquires a SAML assertion for the user, including the current role and context, from an identity provider. They refer to this process as single sign-on (SSO), but, in line with established terminology, login credentials are (only) handled by a trusted identity provider when performing SSO [8].

The assertion is then added to the headers calls to web services. The services invoked check the user's permission based on their own policy. Our approach is more realistic with respect to how BPs acquire identity information and credentials, because we adhere to widely-used protocols specified to this end. It it also more generic because we cover the orchestration of identity-based services, while [18] uses identity information only for access control. In addition, unlike [18], our approach has the important features of addressing *dynamic* authorization constraints (BoD/SoD) (while the static versions are also possible) and performing access control based on attributes asserted by the IdP of the user. We accomplish this by leveraging FIM features and technology systematically.

The draft *WS-BPEL Extension for People* (BPEL4People) specifies the integration of human activities (here called *people activities*) into business processes. It defines authorization only in an abstract way. Users eligible for a people activity are determined using so-called *people queries*. The specification does not define the format of people queries. History-based constraints can be implemented manually in BPEL. BPEL4People

does not provide any support for policy-based specification and enforcement of such constraints.

Bertino et al. [4, 5] describe the RBAC-WS-BPEL authorization model. The model consists of a role hierarchy, permissions to execute activities, an 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. For the constraints, a special language called 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 the two different kinds of policies. BPCL supports arbitrary predicates as constraints. However, evaluation of the predicates is not addressed. The architecture does not address the integration with identity management.

Paci et al. [40] 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. We consider additional identity-management features, such as web-service calls using identity information. In contrast to [40], we only consider privacy issues arising from the use of identifiers, but not issues caused by using attributes. Concerning authorization, the approach could be integrated into our architecture. However, [40] does not suffice when attribute values are used for other, application-specific purposes. This is because the solution in [40] only considers attributes used in authorization policies.

9 Conclusions and Future Work

We have motivated and described an architecure that combines federated identity management and business-process management, based on standards in the two domains. We have described how this architecure is embedded in the overall architecture of a trust network. Finally, we have explained our implementation of this architecture using the ZXID library and other open-source components. Hence, we offer native support for identity BPs. This allows to easily create applications that adapt their functionality to the individual user. Our case study shows that the functionality provided by our BPMS would be hard to implement manually.

Future work will address mechanisms that provide more privacy and allow more user control over disclosure of identity information, e. g., filtering identity information sent to third parties based on process- and user-specific settings. We will also investigate how one can provide identity information to BP instances.

Acknowledgement

The research leading to these results has received funding from the Seventh Framework Programme of the European Union (FP7/2007-2013) under grant agreement n^{o} 216287 (TAS^3 - Trusted Architecture for Securely Shared Services).

References

[1] Modular PERMIS Project. http://sec.cs.kent.ac.uk/permis/.

[2] OpenSAML website. https://wiki.shibboleth.net/confluence/display/OpenSAML/.

[3] ZXID website. http://www.zxid.org.

[4] Elisa Bertino, Jason Crampton, and Federica Paci. Access Control and Authorization Constraints for WS-BPEL. In *Proceedings of the IEEE International Conference on Web Services*, pages 275–284, 2006.

[5] Elisa Bertino, Lorenzo Martino, Federica Paci, and Anna Squicciarini. *Security for Web Services and Service-Oriented Architectures*. Springer, 2009.

[6] Conor Cahill. Liberty Open Source Toolkit.

[7] Kim Cameron. The Laws of Identity. www.identityblog.com/?p=352, 2006.

[8] David Chadwick. Federated Identity Management. In *Foundations of Security Analysis and Design V*, pages 96–120. 2009.

[9] David W. Chadwick. *Securing Web Services: Practical Usage of Standards and Specifications*, chapter Dynamic Delegation of Authority in Web Services. IGI Global, 2008.

[10] David W. Chadwick and Kaniz Fatema. A privacy preserving authorisation system for the cloud. *J. Comput. Syst. Sci.*, 78(5):1359–1373, September 2012.

[11] D.W. Chadwick, Wensheng Xu, S. Otenko, R. Laborde, and B. Nasser. Multi-session separation of duties (msod) for rbac. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 744–753, 2007.

[12] Anis Charfi, Benjamin Schmeling, Andreas Heizenreder, and Mira Mezini. Reliable, Secure, and Transacted Web Service Compositions with AO4BPEL. In *Proceedings of the European Conference on Web Services*, pages 23–34, 2006.

[13] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, World Wide Web Consortium, [Online], 15 March 2001.

[14] Thomas Erl. *Service-Oriented Architecture – Concepts, Technology, and Design*. Pearson Education, 2005.

[15] Heiko Görig. Context-based Access Control for BPEL (Engines). Diplomarbeit, Universität Stuttgart, 2009. (in German).

[16] David Hollingsworth. *Workflow Handbook 2004*, volume 10, chapter The Workflow Reference Model 10 Years On.

[17] David Hollingsworth. The Workflow Reference Model. WfMC Specification TC00-1003, Workflow Management Coalition, 1995.

[18] Waldemar Hummer, Patrick Gaubatz, Mark Strembeck, Uwe Zdun, and Schahram Dustdar. An integrated approach for identity and access management in a SOA context. SACMAT '11, pages 21–30, 2011.

[19] K. Böhm et al. A Flexible Architecture for Privacy-Aware Trust Management. *Journal of theoretical and applied electronic commerce research*, 5, 2010.

[20] Ramarao Kanneganti and Prasad Chodavarapu. *SOA Security*. Manning Publications Co., 2008.

[21] D. Richard Kuhn and David F. Ferraiolo. Role-Based Access Control (RBAC): Features and Motivations. 1995.

[22] Liberty Alliance Project. Liberty ID-WSF Security Mechanisms Core Version 2.0, 2006.

[23] Liberty Alliance Project. Liberty ID-WSF Security Mechanisms Core (Version 2.0), 2006.

[24] Liberty Alliance Project. Liberty ID-WSF Web Services Framework Overview Version 2.0, 2006.

[25] Liberty Alliance Project. Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification, 2007.

[26] Liberty Alliance Project. Liberty ID-WSF Discovery Service Specification, 2007.

[27] Eve Maler and Drummond Reed. The Venn of Identity: Options and Issues in Federated Identity Management. *IEEE Security and Privacy*, 6:16–23, 2008.

[28] M. Menzel, I. Thomas, and C. Meinel. Security requirements specification in service-oriented business process management. In *ARES '09*.

[29] Jens Müller and Klemens Böhm. The Architecture of a Secure Business-Process-Management System in Service-Oriented Environments. In *ECOWS 2011*.

[30] OASIS. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, 2005.

[31] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard, February 2005.

[32] OASIS. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, 2005.

[33] OASIS. SAML 2.0 profile of XACML v2.0. OASIS Standard, 2005.

[34] OASIS. Web Services Security: SAML Token Profile 1.1. OASIS Standard, 2006.

[35] OASIS. Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). OASIS Standard, 2006.

[36] OASIS. Web Services Business Process Execution Language Version 2.0, 2007.

[37] OASIS. WS-SecurityPolicy 1.2. OASIS Standard, 2007.

[38] OASIS. Security Assertion Markup Language (SAML) V2.0 Technical Overview, 2008.

[39] OpenID Foundation. OpenID Authentication 2.0, 2007.

[40] 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.

[41] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, and Identity Management. Technical report, August 2010. v0.34.

[42] 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, 2007.

[43] Kaijun Tan, Jason Crampton, and Carl A. Gunter. The Consistency of Task-Based Authorization Constraints in Workflow Systems. *17th Computer Security Foundations Workshop, IEEE*, 2004.

[44] Christian Wolter and Andreas Schaad. Modeling of Task-Based Authorization Constraints in BPMN. In *Business Process Management*, 2007.

[45] World-Wide Web Consortium. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 2007.

[46] Eric Yuan and Jin Tong. Attributed Based Access Control (ABAC) for Web Services. In *ICWS 2005*, pages 561–569, Los Alamitos, CA, USA, 2005.

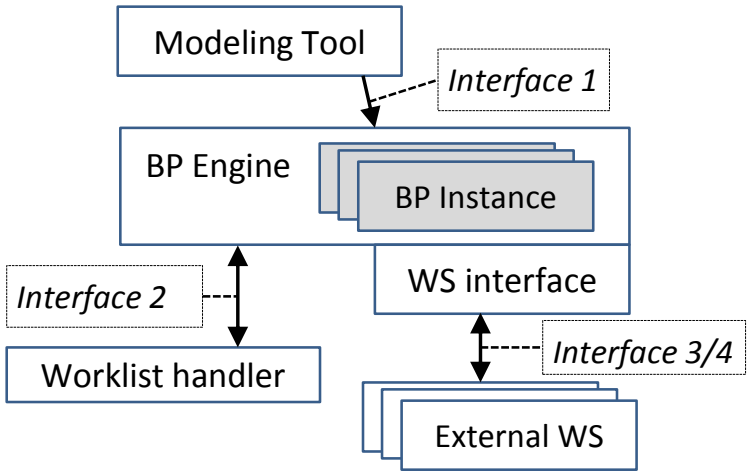

Figure 1 Basic Architecture of a BPMS

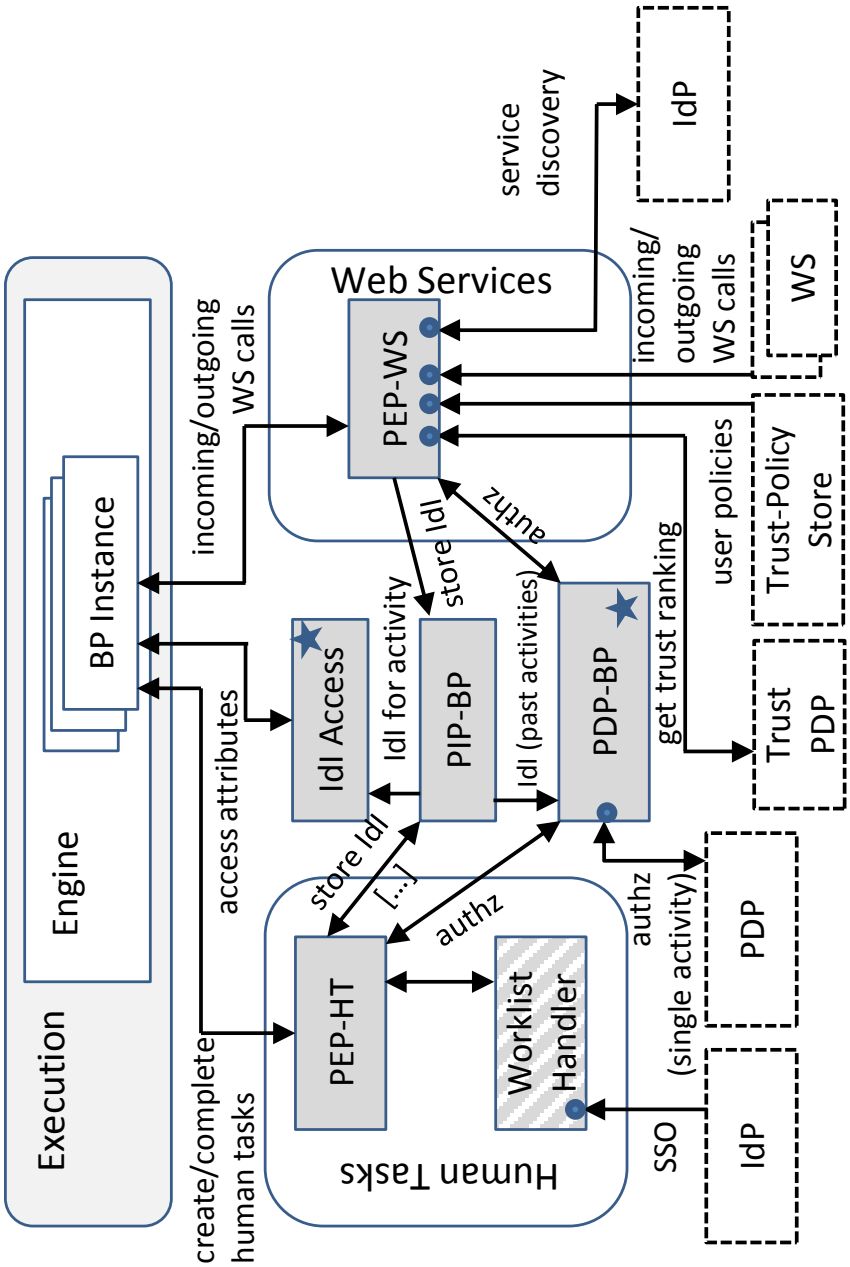

Figure 2 Architecture of a BPMS with FIM Support

A Performing Correlation for Incoming WS Calls

In Subsection 5.5.3, we raised the question how the PEP-WS can determine the BP instance and activity an incoming WS call is directed to. This is necessary so that the PEP-WS can

Algorithm 1 PEP-WS performing a call to an identity WS

Require: activity, instanceId, svctype, payload
$identity \leftarrow PDPBP.getIdentity(activity, instanceId)$
$services \leftarrow Discovery.getServices(svctype, identity)$
$maxScore \leftarrow -\infty$
$bestService \leftarrow null$
$policy \leftarrow PolicyStore.getPolicy(identity)$
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 $error$
$result \leftarrow bestService.call(payload)$

perform authorization before the message is actually processed by the BP instance. We sketched a way to accomplish this, and now describe it in more detail.

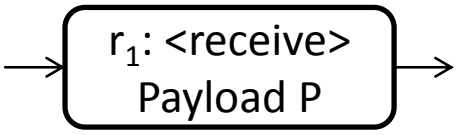

Figure 3 Original <receive> activity

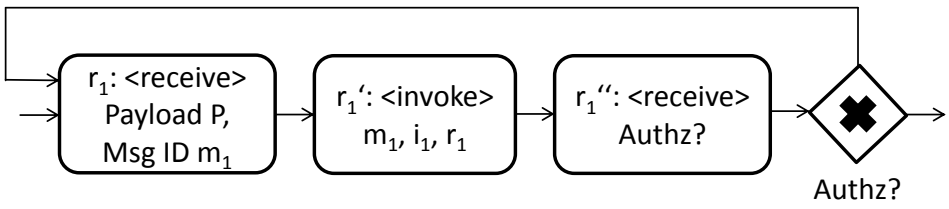

Figure 4 Fragment inserted for the <receive> activity

Consider a process with a <receive> activity r_1, which receives a message with payload P (Figure 3). For each such activity, the BP 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.
- After r_1, an <invoke> activity r_1' is inserted, which calls the PEP-WS to inform it about the BP 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 authorization decision from the PEP-WS. If authorization is granted, execution of the BP instance continues normally. If not, execution of r_1 starts again.

Figure 4 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="sendAuthzDecision"
    variable="authzDecision" />
  </sequence>
  <condition>
   <!-- authz granted -->
  </condition>
 </repeatUntil>

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