Karlsruhe Reports in Informatics 2011,28
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Using Federated Identity Management in a Business-Process-Management System – Requirements, Architecture, and Implementation

Jens Müller and Klemens Böhm

2011

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Using Federated Identity Management in a Business-Process-Management System – Requirements, Architecture, and Implementation

Jens Müller and Klemens Böhm

Information Systems Group,
Institute for Program Structures and Data Organization,
Faculty of Informatics,
Karlsruhe Institute of Technology

Abstract. Identity management is a key component of information-system security. In the context of service-oriented architectures (SOA), federated identity management (FIM) is important. Nowadays, business-process management (BPM) is used for diverse applications to orchestrate activities of web services and humans in SOA. Involving humans in business processes implies a notion of identity. Nevertheless, the relationship between FIM and BPM has not been systematically examined until now. We perform such an analysis, which yields the characteristics of FIM concepts in BPM. Based on standards for BPM, access control and FIM, we propose an architecture of a BPM system with FIM support and discuss design alternatives. The system implements FIM concepts based on declarative configuration, 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.

1 Introduction

Identity is "a property of a subject that enables it to be identifiable and to link items of interest to the subject" [27]. *Digital identity* refers to attribute values attributed to an individual which 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 information-system security, for authentication and access control in particular. An identity-management system has to check the identifiers and attribute values that a subject claims to possess, i.e., authenticate the identity of the subject. On this basis, a system can decide whether to grant the subject access to some resource.

Federated identity management (FIM) is a set of technologies and processes that let computer systems distribute identity information dynamically and delegate IdM functionality to other systems [15]. A FIM 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 [3].

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 the Security Assertion Markup Language (SAML) [25]. However, there are no established concepts for higher-level applications of FIM, such as in policy management or in business processes (BPs).

Business-process management (BPM) is well-suited to orchestrate the behavior of loosely-coupled systems in service-oriented architectures (SOA). Common BPs involve services as well as human users, implying a notion of identity. Authorization constraints restrict who is allowed to perform activities based on this notion and a relationship between activities. Separation of duty (SoD) [8] requires different users for two or more activities. With binding of duty (BoD) [28], the same user has to perform several activities. A BPM system (BPMS) requires authentic attributes of users: First, it needs to decide whether a user may perform a specific actitivity. Second, BPs dealing with personally-identifiable information (PII) cannot guarantee correct results without authentic input. They also depend on user preferences. E.g., a user might have specified which data he is willing to disclose to applications of a specific kind. When a process can access such preferences, it can avoid bothering the user unnecessarily. It is important to note that BPM is a generic technology with diverse applications. Individual process definitions will use the features of a BPMS differently, and the BPMS has to support this.

Integrating FIM support into a BPMS facilitates what we call *identity business processes*, allowing developers of BP applications to easily use the services available in an IdM federation. The necessary configuration could be very lightweight, and the application developer can provide it as annotations to the process model [16]. For example, to use FIM for access control, he needs to state the attribute values required from users to perform an activity of the process. In return, users can use SSO to log into the user interface, and the system automatically authenticates the attributes of the user and compares them to the required ones. Further, users have a more consistent experience when using different service providers: They can easily re-use existing accounts and provide access to their PII to applications automatically. In addition, applications can respect their personal 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 extend the conventional architecture of a BPMS to support FIM concepts? How can advanced BPMS functionality in SOA benefit from FIM? Which configuration is needed to customize the implementation of FIM features to different BP definitions?

All these problems are challenging. First, it is unclear how FIM fits into the architecture of a secure BPMS: Which interfaces of the classical BPMS architecture [5] 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, and allow execution of existing process definitions with 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. Fourth, the resulting BPMS must maintain user privacy. To this end, it must support the respective FIM features, like identity mapping between pseudonyms, prevent leaking of identity information between process instances, and increase user control over the disclosure of their PII.

Our contributions now are as follows:

- We analyze how FIM concepts can be used in BPs, and describe peculiarities that arise when combining them with BPM concepts.
- We describe architecture extensions of a BPMS that allow for secure process execution in SOA with FIM.
- We describe how to make BPM more user-centric by addressing individual user preferences using FIM (e. g., personalized trust policies).
- We say how to implement the extensions using the open-source ZXID library.

This report is structured as follows: We explain fundamental concepts and technologies in Section 2. We then analyze the requirements on our system (Section 3), followed by the actual system design (Section 4). We then briefly explain our implementation in Section 5. Section 6 concludes.

2 Fundamentals and Related Work

2.1 Business-process management in SOA

The WfMC reference model [5] defines an architecture for BPMS, consisting of several interfaces. The central component is the BP execution engine (*engine*). Interface 1 facilitates the deployment of process definitions. This includes the security configuration of process definitions. Interface 2 handles interactions of the BPMS with users via *human tasks* through the *worklist handler*. Interfaces 3 and 4 handle the interaction with applications and other BPMS. Interface 5 deals with administration and monitoring. In SOA, process definitions commonly use WS-BPEL [24]. Applications and other processes are provided as web services using SOAP [29].

Next to accomplishing application functionality, a BPMS has to deal with the challenges of a SOA concerning security and dynamism: Services used in processes often need to be selected at run-time. When dealing with PII, users should be able to influence which service is selected. It should be easy for application developers to specify this. For encrypted connections, it is necessary to acquire the public keys of the services invoked. Access control for human tasks

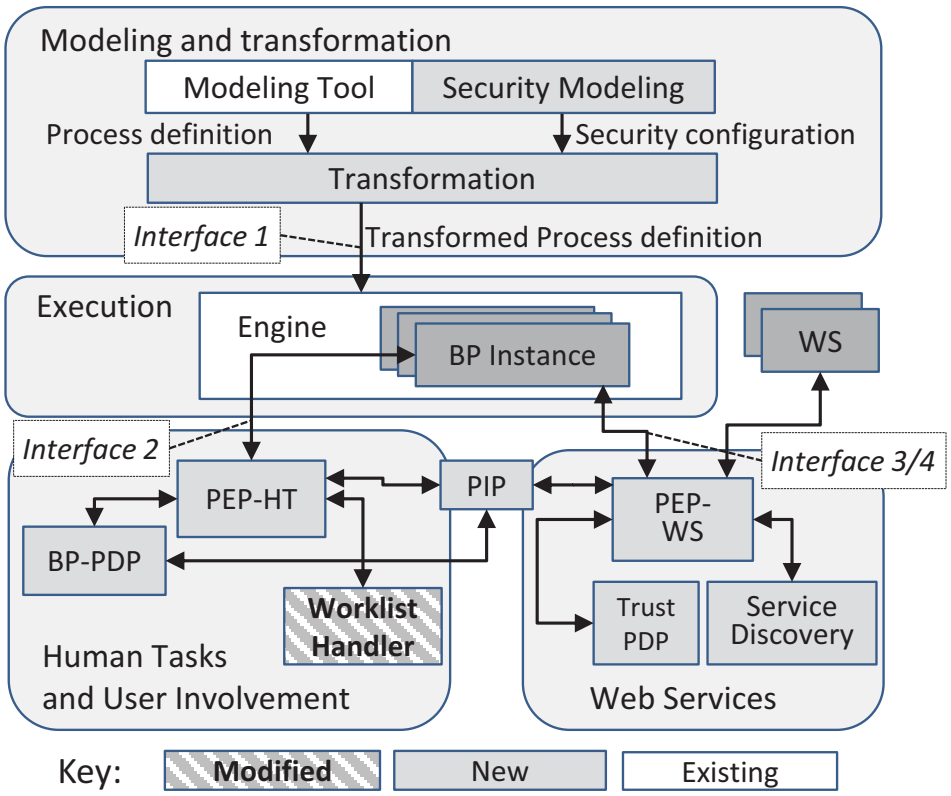

Fig. 1. Architecure of a BPMS with security extensions

is based on the state of process instances (namely constraints involving several tasks) and a policy specific to the process model. We also have found that some security-related interactions with users are frequent, such as agreeing to terms of service, or selecting a service provider. [16] gives a systematic overview of such interactions, called *user involvements*.

To deal with these additional requirements, we have proposed architecture extensions to the WfMC reference model [17]. Figure 1 is a simplified diagram of this architecture. The important changes are: (1) Communication of the engine with the worklist handler and with web services is routed through proxies, called *Policy Enforcement Point for Human Tasks* (PEP-HT) and *Policy Enforcement Point for Web Services* (PEP-WS), respectively. The PEP-HT persists the context a human task was created in and handles access-control requests relating to human tasks. The PEP-WS routes calls to the correct service, checks its trustworthiness and applies encryption and cryptographic signatures. (2) Process definitions are instrumented so that they send web-service calls or human tasks to the PEP-WS/PEP-HT, including appropriate context (i. e., which activity in which process instance has caused the request). (3) The PEP-WS invokes a service-discovery component to find available services and checks their trustworthiness by sending a request to a Trust PDP. (4) The Business-Process Policy

Decision Point (BP-PDP) takes access-control decisions enforced by the PEP-HT, and the Policy Information Point (PIP) stores the history needed for these decisions.

2.2 Identity-management concepts

In this section, we introduce FIM concepts. In Section 3, we analyze their impact on BPM in detail.

Identity governance Governance tasks in FIM comprise (a) establishing the relationship between users and identity providers (IdP), i.e., creating accounts, verifying user identities by some real-world mechanism and providing credentials, and (b) establishing a federation between an IdP and a service provider (SP), which requires creating a trust relationship and respective configuration on both sides. In what follows, we assume that these relationships exist.

Providing identity information to SPs In FIM, an IdP (or asserting party) asserts the correctness of identity information about a subject (user). 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 web-based worklist handler, he needs to authenticate. As users access web-based services at many different SPs, 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 SP. 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 SPs.

Types of identity information There are different 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 SP (e. g., the type of driver's license the user holds). They provide a more fine-grained alternative to roles. The IdP asserts that this information is authentic.

Purpose of identity information Identity information is used for various purposes. 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, offer services available only to a certain age group, or providing information relevant to the user's location. Yet another purpose is authorization: Using attributes for this purpose is known as *attribute-based access control* (ABAC) [30]. ABAC is a generalization of role-based access control. It also supports attributes for objects, actions, and the environment. It is the underlying paradigm of XACML [19].

Pseudonyms Identifiers valid only in a specific context are known as pseudonyms. They can be different for each SP, but otherwise be persistent or even change for each login session. Interactions between SPs require a mapping between the different pseudonyms used at each SP.

Identity-based services The functionality of an *identity web service* depends on whom it provides a service for. This includes: (a) Services storing information about the holder of an identity (user). These services are able to answer requests for such information. (b) Services that interact with the user and return his decision. (c) Services that can take a decision based on instructions from a user. In a secure BPMS, use cases include 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.

Acting for users Services often depend on information provided by other services. This is also true for identity web services. This means that they have to call other identity web services on behalf of the identity that has invoked them.

2.3 Technical FIM specifications

We now briefly introduce technical specifications implementing the concepts from the previous section. This should also be useful as a reading guide to these specifications.

SAML The fundamental data structure in the Security Assertion Markup Language (SAML) [26] are assertions [18], which "carry statements about a principal that an asserting party claims to be true". SAML defines different kinds of statements. 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, which contains a `NameID` element. [18] defines different kinds of `NameID`s. Core alternatives are globally unique names like e-mail addresses and X.509 subject names, privacy-preserving persistent identifiers, which have no correspondence to an actual identifier and are specific to a given service provider, and transient identifiers, i. e., random and temporary values.

[18] also defines protocols, i. e., exchanges of protocol messages, which contains assertion in most cases. Examples are the *Assertion Query and Request Protocol* or the *Authentication Request Protocol*. Note that protocols only define messages, while bindings specify a particular transport mechanism, e. g., HTTP POST. Profiles [20] define in more detail how to use SAML for a particular application, providing for better interoperability between different implementations. There are two types of profiles. One contains a set of rules describing how to embed SAML assertions into or extract them from a framework or protocol. The other one describes how to use SAML features in a particular context by specifying further details left open in the core specification. An important example

is the *Web Browser SSO Profile*, which implements the Authentication Request Protocol using the HTTP Redirect, HTTP POST and HTTP Artifact bindings. It allows to transfer identity information from an IdP to a web frontend of an SP.

The SAML Token Profile of WS-Security [22] allows using SAML assertions as tokens in WS-Security [23] headers of SOAP messages. The SP receiving the message has to validate the evidence provided by the caller according to the confirmation method specified in the assertion (e. g., that the sender holds a specific key).

ID-WSF The Liberty Alliance has developed ID-WSF [11], a framework for identity web services. The SOAP binding [14] provides an invocation framework for identity services. It defines technical details like SOAP headers and status codes. The ID-WSF Security Mechanisms specification [10] defines the use of tokens for message authentication, including tokens that specify the invoking identity. [9] specifies how to use SAML assertions as authentication tokens. In addition, ID-WSF defines an SSO Service that allows a system to obtain SAML assertions as security tokens. It also defines an Identity Mapping Service that translates references to users into alternative formats or identifier namespaces [12]. The Discovery Service specification [13] 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.

XACML XACML [19] is a language for access-control policies. It includes a reference architecture and format for decision requests and results. The SAML 2.0 Profile of XACML 2.0 [21] defines an extension of XACML authorization-decision queries, so that SAML attributes can be transmitted to the PDP. This allows to use identity information for authorization directly.

2.4 Related work

We are not aware of any work systematically combining BPM and FIM, except for [4]. It proposes extensions for a BPEL engine to let processes access the content of security tokens. This approach only considers web-service interactions and requires explicit activities accessing 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, [4] is one possibility to give processes access to profile information contained in tokens (see **F11** in Section 4.1).

3 Requirements Analysis

This section addresses two issues: First, we explain how some general requirements for FIM applications are relevant for BPM. Second, we analyze the characteristics individual FIM concepts exhibit in BPM and the resulting requirements.

3.1 General requirements

The *laws of identity* [2] are recommendations for FIM implementations, mainly concerning interoperability, privacy, and user control. They are the result of intensive discussions within the identity-management community. We now derive requirements on the implementation of FIM features in a secure BPMS.

One law requires that the system "must only reveal information identifying a user with the user's consent". Regarding BPM, this applies to process instances and third-party services in particular. A BPMS should control disclosure of identifying information to them. Another law requires "unidirectional" identifiers valid only for one SP to prevent combination of identity information provided to different parties. In BPM, this means isolation between process instances. Third, there should be a pluralism of operators and technologies, and "multiple identity systems run by multiple identity providers" should be supported. Our solution will use standard FIM technologies (cf. Section 2.3) and can interoperate with multiple identity providers. The architecture should be based on generic concepts and their application to BPM, to be independent of the concrete technology. Finally, the identity system should provide "unambiguous human-machine communication mechanisms offering protection against identity attacks". A BPMS helps by providing the same authentication mechanisms for different kinds of applications. The BPMS should also allow re-using predefined interactions for, say, giving consent, choosing service providers information is disclosed to, etc.

3.2 Individual FIM concepts

BPM is a generic technology for different kinds of applications. Hence, a BPMS needs to support different security concepts, including identity management, even though not every application will use all of them. In addition, the mechanisms provided by the BPMS must be configurable to fulfill the security requirements of different applications.

In the following, we develop requirements on the integration of different FIM concepts into a BPMS. First, we explore the relationship of these concepts to activities and other entities, such as users or external services, related to BP instances. This affects how the BPMS needs to process identity information at run time. Second, we examine how the requirements of different applications regarding a concept can differ. We look at how a respective configuration can be expressed based on the elements of process definitions. This allows us to decide which mechanisms are needed in the BPMS to support FIM, and which instance-specific data the BPMS needs consider. This forms the basis for mapping the required functionality to an architecture and its implementation.

The list of FIM concepts considered here is based on the one introduced in Section 2.2, but we structure the items differently, taking their importance for BPM into account.

Governance of the FIM network The only governance task involving the BPMS is the establishment of a federation between the BPMS and the IdPs to be used. This does not affect individual processes and thus does not require any configuration.

SSO In a BPMS, users interact with the worklist handler through a web-based interface to perform human tasks. To authenticate to the worklist handler, they use SSO. The handler needs to store identity information for the duration of the respective SSO sessions. When a user tries to perform a task, the information is used for authorization. When the user has performed a task, the identity information is stored, remembering the relation to the activity and process instance in question. We do not see any need, however, to configure how to perform SSO.

Incoming identity-WS calls The BPMS may also receive web-service calls containing identity information. In this case, authorization for the message can happen immediately. If successful, the identity information is connected to the activity and process instance receiving the call. Again, there is no need to configure the acquisition process itself.

ABAC ABAC uses attributes of users to decide whether access on some entity should be granted. In BPM, this concerns human tasks as well as service calls directed to process instances in the name of a user. The access-control decision can be taken for each such process activity individually. The attributes required depend on the application. Thus, BP definitions must specifiy the attributes needed for each task.

History-based constraints BPs are potentially long-running, stateful, and can involve several users performing tasks. Authorization needs to take the relationship between tasks into account. A typical SoD constraint requires two different users for the *Authorize payment* and *Issue cheque* activities. To this end, the BPMS must remember for each BP instance who has performed the tasks. Pseudonyms must be persistent, otherwise constraints cannot be correctly enforced. Application developers must specify constraints for each BP definition.

Using attributes for a personalized service Business processes coordinate web services and human activities. In particular, they compose web services providing services for a specific user. Depending on the application, the user's attributes 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. Such customizations take the connection of identified users with process instances into account, and of course their identity information. However, the actual mechanisms used highly depend on the individual application. The task of the BPMS solely is to provide user attributes to BP

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

Acting on behalf of users When BPs invoke identity web services, they do so on behalf of a user. This means that the BPMS needs to store identity information acquired by activities to use it for outgoing calls. 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, the source of identity information must be specified. Again, this can be accomplished by referring to an activity and the identity information of the user who has performed it.

3.3 Summary

We have explored the relationship between requirements on FIM applications and FIM concepts on the one hand, and BPM on the other hand. The latter has resulted in an assessment of the run-time context and of the configuration needed to implement the features.

4 System Design

In this section, we derive the structure of a BPMS for identity BPs, i. e., its components, their functionality, and the interactions between them. In Section 3.2, we have analyzed the FIM concepts and their characteristics relevant for BPM. These are the features our system has to provide. In particular, we have outlined how the context of BP instances influences the FIM implementation, and where configuration is needed. In addition, the entire system should fulfill general requirements on FIM applications, as introduced in Section 3.1, namely: The system should disclose PII only with user consent, isolate BP instances against correlation of identity information, provide interoperability and support competing FIM technologies, and ensure unambiguous human-machine interaction. Finally, the following design goals are important, and ease implementation and system maintenance:

D1 *Traceability:* We want to assign responsibility for a security-relevant function to one component. This reduces complexity and makes it easier to guarantee correct behavior of the system.

D2 *Abstraction from technologies:* We need to distinguish between a concrete implementation and the generic FIM concepts and 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 replace them when necessary.

D3 *Standards-based architecture:* The architecture of our system and its interactions with the environment should be based on established standard architectures and technologies. This includes the WfMC workflow reference model [5], the reference architecture of XACML [19], and the use cases of the SAML profiles [25].

D4 *Declarative configuration:* As mentioned in Section 3.2, the system must be configurable to support the security requirements of different applications. In principle, there are two alternatives: (1) The system provides implementations of the concepts, with operations that allow an application BP to set configuration options based on its internal state. This requires explicit, imperative-style code in the application. (2) Definitions of application BP include declarative-style configurations. The system adapts to the requirements of the application by evaluating the configuration, taking the context 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), so that a transformation component can derive the configuration from annotations of process models. However, when application logic is closely combined with FIM functionality, the imperative approach is inevitable. In summary, we want to provide declarative configuration wherever possible, and interfaces abstracting from technical details otherwise.

4.1 Individual design decisions

When extending the architecture of a BPMS with the required functionality, it is not always clear how to do this. This is because there are alternative solutions that fulfill the design goals to a different degree. We now go through different areas of functionality. We identify and describe design alternatives, discuss them, and decide which one to pursue.

Identity information F1 *Representation of identity information*: Problem: We need to decide on a format for representing identity information when storing it or transferring it between components of the system. Such information consists of SAML assertions (transmitted as security tokens attached to WS calls). Several components exchange identity information: The worklist handler and the PEP-WS acquire it. The PIP stores it, and the BP-PDP uses it for access-control decisions. Options: (1) Always transfer complete blocks of identity information. If a component needs a particular piece of it, it would extract it itself. (2) Store the information centrally and only pass references. The storage layer provides an API to access individual pieces of identity information. Discussion: The format of the information is complex and depends on the technology used. This is an advantage of Option (2) regarding **D2**. Defining and implementing an API requires additional effort, but this effort also makes it easier to achieve **D2**. Conclusion: We will use a storage layer for identity information.

F2 *SSO:* Problem: Users must be able to authenticate to web interfaces of the BPMS. In the reference architecture, the worklist handler provides the only

web interface. With FIM, the technique applicable for authentication in a web interface is single sign-on (SSO), involving interaction with an IdP. The worklist handler must be able to relate the identity information acquired through SSO to any further activity of the user in the session. Conclusion: In **F1** we have decided to use a storage layer for identity information. Consequently, the worklist handler stores the information there. We can easily accomplish re-identification using session cookies.

Access control F3 *Access control in general*: Problem: Activities in BP instances can be triggered externally from two sources: Human tasks and incoming web-service calls. Both are subject to access control. To this end, identity information about users, the context of the activity (i. e., which activity in which BP instance is triggered?), and the history of process instances is required. The BPMS has to collect this information and take a decision. Conclusion: We will use a central component for policy decisions, the BP-PDP, and store the history in the PIP. This allows constraints involving both human tasks and incoming calls.

F4 *Policy evaluation*: Problem: Our access-control policies have two parts: First, there are conditions defining when users may perform activities. Such conditions may refer to required attribute values or to the IdP that has asserted the information. Their evaluation is stateless. Second, there are constraints based on the relationship between activities in a BP instance, e. g., BoD and SoD. Such constraints restrict access rights, and their evaluation requires history information. Options: We have to decide whether to implement a monolithic BP-PDP or a modular one, with separate sub-components responsible for the two different parts of the policy. Discussion: On the one hand, a monolithic PDP might allow some optimizations. 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. Being able to plug in an existing implementation is favorable regarding **D2**. Conclusion: We choose a modular implementation, because we deem a flexible architecture more important. For the stateless part, we propose using an existing PDP implementation through the SAML profile for XACML [21].

F5 *Authorization constraints and pseudonyms*: Problem: A problem arises when a user has several identities, and the BPMS does not know this. Then one can evade SoD constraints. A similar problem exists for transient pseudonyms only valid for a limited time. As BPs often last longer, a user might access the system with different transient identifiers. Options: We see two alternatives to deal with users who have more than one identity. (1) We can require users to be authenticated by the same IdP for activities that are part of an SoD constraint. For example, the users performing *Authorize payment* and *Issue cheque* would have to be authenticated by the corporate IdP. In addition, the IdP must not allow multiple identities for a person. (2) We can design a federation of IdPs that asserts that two identities are different. For transient identifiers, we see the following alternatives: First, we could disallow the use of transient identifiers when that kind of constraint is involved. Second, IdPs could provide a mecha-

nism asserting the holders of two transient identifiers to be different. Discussion: Requiring a specific IdP restricts the functionality. However, this will not be relevant in most cases. Consider the example from above: Both users are employees of the same corporation, so requiring that they be authenticated by the corporate IdP is reasonable. A mechanism involving multiple IdPs requires substantial modifications of existing protocols. Allowing a comparison of transient identifiers would require changes in the IdP, and we deem it incompatible with the purpose of transient identifiers. In contrast, we do not see any problems in disallowing transient identifiers when they cause problems. Conclusion: In summary, we require a specific IdP and disallow transient identifiers when needed.

F6 *Access control in the worklist handler*: Problem: Users are logged into the worklist handler for some time. They view their worklist and perform tasks. The handler needs to get a decision from the BP-PDP whether to include a task in the worklist, or to allow a user to perform it. A user can also claim a task for himself without immediately finishing it. This affects the evaluation of constraints: For instance, no other user may claim or finish another task when BoD holds for the two tasks. We need to specify how the components interact to perform authorization and to have the necessary history available. Conclusion: Most of this specification is rather obvious, given the architecture from Section 2.1. The worklist handler requests authorization from the PEP-HT, which can look up task details from the PIP and forward the request to the BP-PDP in turn. When users claim or complete tasks, the worklist handler informs the PEP-HT, which registers this fact in the PIP. However, authorization requests for each task each time the user views his worklist yield poor performance. As a solution, we propose to cache the results for some time.

Web services F7 *Incoming web-service calls*: Problem: A proxy component, the PEP-WS, receives incoming web-service calls. The tasks of the PEP-WS are extracting identity information and making it available to other components, performing access control, and forwarding the message to the BP engine. Some of these points are straightforward: Identity information is handed to the storage layer. Access control is performed by calling the BP-PDP and including the context of the activity invoked. This poses one problem, however: Normally, it is the BP engine that performs *correlation*. Correlation means determining the context of the call, i. e., the activity and BP instance the call is directed to. However, the BP-PDP needs this context to evaluate constraints. Further, the PEP-WS needs it to store the identity information in the PIP. Options: We see several approaches to this problem: (1) Re-implementing correlation in the PEP-WS. (2) Instrumenting the process, activities receiving calls in particular: The inserted fragment would send 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 result as a reply. If authorization is denied, the received message is discarded, and the activity is started again. (3) Closer 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 simpler process instrumentation. Discussion: We deem (1) impractical. It would require re-implementing a substantial part of the functionality of the engine in the PEP-WS. To accomplish this, the PEP-WS needs 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 is easiest to implement. Conclusion: We start with (4) and leave (2) as future work.

F8 *Service discovery and selection*: Problem: For outgoing calls, i.e., from the BPs to external services, the BPMS needs to know which service to call. In many cases, it must choose this service at run time, because user preferences have to be considered. In detail, it needs to look up available services, choose eligible ones according to the user's trust policy, and ask the user for the final choice. Finally, the service selected is stored in the PIP. Options: We see two alternatives each (1) to deal with personalized trust policies and (2) to involve the user. (1a) The component that stores the policies (policy store, PS) and Trust PDP are separate components. The BPMS retrieves the trust policy from the PS and uses it to invoke the Trust PDP. (1b) The Trust PDP knows the trust policies of users. Note that it depends on the user which Trust PDP should be used. This means that discovery has to be performed. (2a) A separate UI component can be used. (2b) The worklist handler can present tasks for service selection in the same way as application tasks, using pre-defined process fragments called *user involvements*. Annotations to the process model result in the inclusion of such fragments, cf. [16]. Discussion: (1a) discloses the trust policy to the BPMS, (1b) does not. In addition, (1b) is more interoperable, because there currently is no common trust-policy language in widespread use. An argument in favor of (1a) is that the user's Trust PDP is not necessarily able to compute a trust ranking for all services available. Regarding (2), (2b) is more natural to express sequences of activities embedded in the application process. Conclusion: For (1), no alternative is clearly superior. Both depend on the availability of respective services for users. Our current implementation is similar to (1a). To involve the user, we choose (2b).

F9 *Outgoing web-service calls*: Problem: This functionality requires calling the correct service and route the reply back to the process instance. The PEP-WS has to use the correct protocol and to perform identity mapping, i. e., get tokens with pseudonyms valid for the SP called. To do so, it has to decide which identity to use. Options: Using the ID-WSF protocols and performing identity mapping is a straightforward implementation task. Alternative ways exist to choose the correct identity: (1) The BPMS can perform this choice based on a policy and on the history of the process instance. The policy specifies another process activity performed before the call. The token, part of the identity information acquired in this activity, is used for the call. (2) The process itself can make this choice explicitly: When the process invokes a service, it specifies the identifier of the token to be used, known from incoming calls or human tasks completed before.

Discussion: (1) is more flexible, but requires process designers to use extended interfaces when receiving calls or invoking services. With (2), the additional configuration needed does not affect the application functionality. Moreover, (2) fulfills **D4**. Conclusion: We decide to use (1). The BP-PDP evaluates the respective policy, using history stored in the PIP, and returns a reference to identity information, which also contains the token.

Miscellaneous F10 *Dealing with targetted identities*: Problem: Pseudonyms are specific to one SP. The goal is to prevent what we call *combination* of identity information about one user that is available to different SPs. As a BPMS can execute different unrelated applications, combination should also be prevented between BP instances. Options: We see three alternatives: (1) Let the IdP create different pseudonyms for each process instance. (2) Do not disclose pseudonymous identifiers to process instances. Instead, perform functions needing them, such as constraint evaluation, in the BPMS. (3) Perform an internal mapping, i. e., let the BPMS provide different but persistent pseudonyms to process instances. Discussion: (1) would require registering each process instance as a SP and establish a trust relationship with the IdP. This is not feasible with current protocols. (3) is quite similar from the perspective of processes, but an efficient implementation is possible without changing existing protocols. (2) is preferable according to **D4**. If the BPMS provides all functionality dealing with identifiers that applications need, there is no additional benefit in having (3) as well. Conclusion: We implement (2), with the option to add (3) later, should we observe that applications use identifiers in ways we cannot easily implement in the BPMS itself.

F11 *Providing profile information to process instances*: Problem: One purpose of attributes is to transfer profile information to an application, cf. Section 2.2. We need ways to do so automatically, but with user control over the disclosure. Options: (1) Users often give personal data to BP instances through human tasks they perform in the worklist handler. We can improve this manual process by using attribute values: The worklist handler could use attribute values as default values of form fields. A configuration determines the mapping between attributes and form fields. (2) We can let processes query the PIP for attribute values. There are different ways to achieve this [4]. We also need an interface that can provide attribute values to BP instances, based on the instance ID and the activity that has acquired the information. Discussion: One advantage of the first way is that it is declarative and requires no changes in process definitions, only in definitons of human tasks. In addition, the user can control information disclosure in an intuitive way. However, the process cannot easily determine whether the information is authentic, and it is not applicable to identity information acquired by incoming calls. (2) is more flexible, but how to support BP developers requires additional exploration. Allowing the user to control disclosure is more challenging with (2) as well. Conclusion: We think that both approaches are useful and can complement each other. Our current implementations allows BP instances

to query attribute values from the PIP, but does not provide for user control yet. It is relatively easy to add (1) to our implementation of the worklist handler.

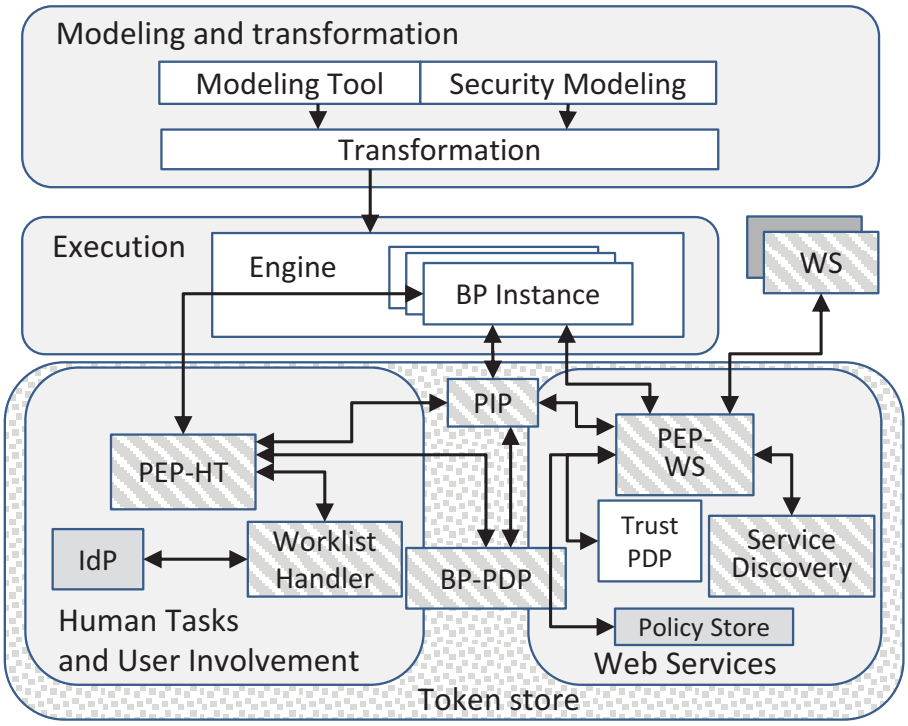

Fig. 2. Architecture of a BPMS with FIM support

4.2 Resulting architecture

Our design decisions result in a refined version (Fig. 2) of the architecture from Section 2.1, with the following modifications: (1) The *Token Store* is introduced as a storage layer for identity information (**F1**). It indexes entries by a unique ID and allows retrieving identifiers, attribute values, and the raw security tokens. (2) Support for SSO is added to the *Worklist handler* (**F2**). This requires an external component, the *IdP*. Authorization queries now refer to IDs of tokens acquired through SSO, instead of user names directly (**F6**). (3) The *PEP-WS* is extended to handle identity-web-service calls (incoming and outgoing), cf. **F7** and **F9**. (4) The *BP-PDP* is split into a stateless and a history-aware part (**F4**). (5) *Service discovery* is enhanced so that it can do user-specific trust evaluations, and a *Policy store* is added (**F8**). (6) The *PIP* is extended with an interface that lets processes query attribute values (**F11**).

4.3 Configuration

Our extended BPMS requires different kinds of configuration for identity BPs. First, there are access-control policies for activities, consisting of a stateless part and additional constraints (cf. **F4**, and the restrictions explained in **F5**). Second, a policy determining the choice of tokens for outgoing calls is needed (**F9**). These policies need to be created for each BP definition, e. g., based on annotations of a process model. They must be deployed to the BP-PDP when the BP definition is deployed to the BP engine.

5 Implementation

We developed the architecture presented in this paper as part of the TAS^3 (Trusted Architecture for Securely Shared Services) project. Security, especially identity management, plays a major role. [6] describes the overall architecture.

The core security architecture [7] is implemented using ZXID [1]. It supports the main protocols and simplifies the handling of identity information. Whenever identity information is aquired, ZXID automatically stores it and assigns a randomly generated ID as a handle. All identity information associated with the handle is persisted on disk. Thus, different components can access it as long as they run on the same machine. Many ZXID functions, e. g. for calling an identity web services or sending an authorization request to a PDP, need identity information. These functions take a handle as a parameter, thus abstracting from the internal structure of that information.

The SSO module of ZXID can be implemented in servlets or other technologies for serving web pages. It handles the entire protocol exchange with the IdP. On successful completion of SSO, it creates a new session. The web-service-provider module provides similar functionality for incoming calls. The web-service-consumer module is the counterpart for outgoing calls. Given a ZXID session, it performs any necessary activity, like service discovery, identity mapping, sending the actual call and validating the result automatically. ZXID also contains helper functions to call XACML PDPs, automatically extracting attributes from a session and including them in the request. It also parses the XACML response, allowing to easily determine whether the request has been granted. In summary, we use ZXID as the implementation-dependent layer and as the *Token store* component. Other components are implemented either as Java servlets or SOAP web services. We use MySQL as a storage backend.

6 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 briefly 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.

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. Finally, the system should be extended to provide identity information to BP instances.

Acknowledgment

This research has received funding from the Seventh Framework Programme of the European Union (FP7/2007-2013) under grant agreement n° 216287 (TAS^3 - Trusted Architecture for Securely Shared Services). The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.

References

1. ZXID website, `http://www.zxid.org`
2. Cameron, K.: The laws of identity. www.identityblog.com/?p=352 (2006)
3. Chadwick, D.: Federated identity management. In: Foundations of Security Analysis and Design V, pp. 96–120 (2009)
4. Görig, H.: Context-based Access Control for BPEL (Engines). Diplomarbeit, Universität Stuttgart (2009), (in German)
5. Hollingsworth, D.: The Workflow Reference Model. WfMC Specification TC00-1003, Workflow Management Coalition (1995)
6. Kellomäki (Ed.), S.: TAS3 Architecture. TAS3 Deliverable 2.1, 1st Iteration (2009)
7. Kellomäki (Ed.), S.: TAS3 Protocols, API, and Concrete Architecture. TAS3 Deliverable 2.4, 1st Iteration (2009)
8. Kuhn, D.R., Ferraiolo, D.F.: Role-Based Access Control (RBAC): Features and Motivations (1995), `http://csrc.nist.gov/rbac/`
9. Liberty Alliance Project: ID-WSF 2.0 SecMech SAML Profile (2006)
10. Liberty Alliance Project: Liberty ID-WSF Security Mechanisms Core Version 2.0 (2006)
11. Liberty Alliance Project: Liberty ID-WSF Web Services Framework Overview Version 2.0 (2006)
12. Liberty Alliance Project: Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification (2007)
13. Liberty Alliance Project: Liberty ID-WSF Discovery Service Specification (2007)
14. Liberty Alliance Project: Liberty ID-WSF SOAP Binding Specification Version 2.0 (2007)
15. Maler, E., Reed, D.: The venn of identity: Options and issues in federated identity management. IEEE Security and Privacy 6, 16–23 (2008)

16. Mülle, J., von Stackelberg, S., Böhm, K.: A Security Language for BPMN Process Models. Karlsruhe Reports in Informatics 2011-9, Karlsruhe Institute of Technology, http://digbib.ubka.uni-karlsruhe.de/volltexte/1000023041
17. Müller, J., Böhm, K.: The Architecture of a Secure Business-Process-Management System in Service-Oriented Environments. In: ECOWS 2011
18. OASIS: Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0 (2005)
19. OASIS: eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard (February 2005)
20. OASIS: Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0 (2005)
21. OASIS: SAML 2.0 profile of XACML v2.0. OASIS Standard (2005)
22. OASIS: Web Services Security: SAML Token Profile 1.1. OASIS Standard (2006)
23. OASIS: Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). OASIS Standard (2006)
24. OASIS: Web Services Business Process Execution Language Version 2.0 (2007)
25. OASIS: Security Assertion Markup Language (SAML) V2.0 Technical Overview (2008)
26. OASIS: Security Assertion Markup Language (SAML) V2.0 Technical Overview (2008)
27. Pfitzmann, A., Hansen, M.: A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, and identity management. Tech. rep. (Aug 2010), v0.34
28. Tan, K., Crampton, J., Gunter, C.A.: The Consistency of Task-Based Authorization Constraints in Workflow Systems. 17th Computer Security Foundations Workshop, IEEE (2004)
29. World-Wide Web Consortium: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) (2007)
30. Yuan, E., Tong, J.: Attributed Based Access Control (ABAC) for Web Services pp. 561–569 (2005)