

# Policy Languages for Digital Identity Management in Federation Systems

Elisa Bertino    Abhilasha Bhargav-Spantzel    Anna C. Squicciarini  
CERIAS and Department of Computer Science, Purdue University  
(bertino,squiccia,bhargav)@cs.purdue.edu

**Abstract**—The goal of service provider federations is to support a controlled method by which distributed organizations can provide services to qualified individuals and manage their identity attributes at an inter-organizational level. In order to achieve secure and controlled sharing of identity information and services, different types of policies need to be enforced. Moreover, it is necessary to record information on interactions among the federation entities in order to make authentication and authorization decisions able to take into account the activity history. To achieve these goals we propose a comprehensive assertion language able to support description of static and dynamic properties of the federation system. The assertions are a powerful means to describe the behavior of the entities interacting in the federation, and to define policies controlling access to services and privacy policies. Precisely, we provide a set of consistency checks that are required to assure correctness in the behavior of the system. We also present a grammar for defining flexible and expressive policies based on these assertions. We also propose a log-based approach for capturing the history of activities within the federation; in our approach, the log is implemented as a set of tables stored at databases at the various organizations in the federation. We illustrate how, by using different types of queries on such tables, security properties of the federation can be verified.

## I. INTRODUCTION

Today a global information infrastructure connects remote parties worldwide through the use of large scale networks, relying on application level protocols and services, such as recent web service technology. Execution of activities in various domains, such as shopping, entertainment, business and scientific collaboration, and at various levels within those contexts, is increasingly based on the use of remote resources and services. The interaction between different remotely-located parties may be (and sometimes should be) based on little knowledge about each other. To support these rich experiences and collaborations, more convenient IT (Information Technology) infrastructures and systems are needed. We expect, for example, that personal preferences and profiles of users be readily available when shopping over the Web, without requiring the users to repeatedly enter them. In such a scenario, digital identity management (IdM) technology is fundamental in customizing user experience, protecting privacy, underpinning accountability in business transactions, and in complying with regulatory controls.

Digital identity can be defined as the digital representation of the information known about a specific individual or organization. As such it encompasses, not only login names (often referred to as *nyms*), but many additional information, referred

to as *identity attributes* or *identifiers*, about users. An emerging approach to address issues, such as interoperability across different domains, related to identity management is based on the notion of *federations* [6], [9]. The goal of federations is to provide users with protected environments to federate identities by the proper management of identity attributes. Federations are usually composed by two main entities: identity providers (IdPs), managing identities of individuals, and service providers (SPs), offering services to registered individuals or users. IdP's and SP's can actually be implemented by same servers. Federations provide a controlled method by which federation members can provide more integrated and complete services to a qualified group of individuals within certain sets of business transactions. By controlling the scope of access to participating sites, by enabling secure, cross-domain transmission of users personal information, federations can make more difficult the perpetration of identity frauds, as well as their frequency, and the potential impact of these frauds.

Federations require a number of different policies to be properly set and updated over time. In particular, relevant policies are the security and privacy policies; they are crucial in order to assure that identity information are strongly protected across federations. A possible categorization of these policies is as follows.

- 1) **Resource Authorization Policies:** A resource authorization policy defines the conditions that a subject needs to satisfy in order to be authorized for the resource the policy is specified for. Here, a *resource* can be either a service by a SP or, more generally, any object accesses to which have to be controlled. Resource authorization policies are specified using assertions, that typically require attributes and/or certificates proving identities and/or properties of the requesting users. Resource authorization policies can also be prioritized. For example, some preliminary authorization policies by a given SP may check, before making other checks, if the user has a valid single sign-on identifier for the federation. From the user's perspective he/she could first validate the service provider federation certificate. The validation of the web service provider certificates by the client is important in order to prevent phishing attacks [1]. Complex resource authorization policies are required for a fine grained access for the party's resources.

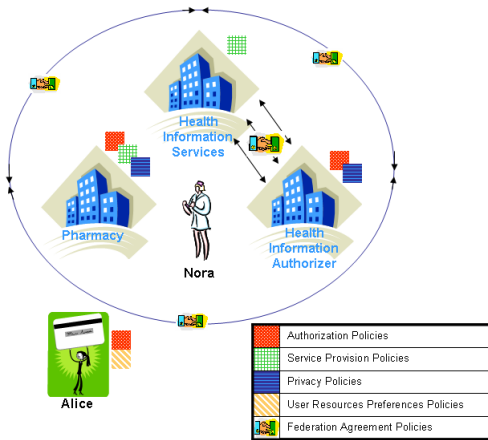


Fig. 1. Illustrating Example 1 and the Federation Policies.

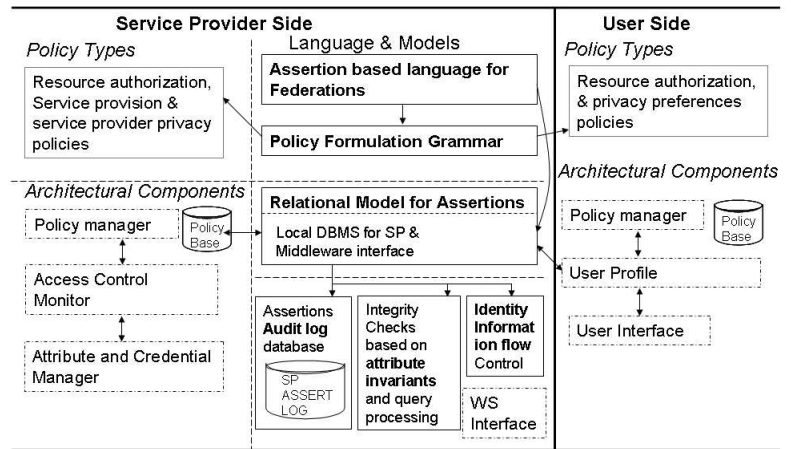


Fig. 2. Architectural Model of the Federation Entities.

Services are the main kind of resources provided by the federation. For each service there could be different authorization policies allowing a subject to qualify for the service. The service that a subject receives may depend on the identity information the subject is willing to disclose to the service provider. For example, if the requester supplies the service provider information about his/her name, address and credit card number, the full version of the service may be provided, as compared to when the requester only gives information about his/her name and address and a thus a trial version of the service is given for a limited time of 30 days. For other types of resources, like attributes and certificates, similar policies can be defined and used during the negotiations. For example before giving the attribute credit card number, the requester should prove it is a member of the Better Business Bureau (BBB).

- 2) **Service Provisioning Policies:** The resource authorization policies for the services may have different ways to authorize a user for a given service. The criteria associated with service provisioning may depend on how the user is qualified for the service. Some of these criteria are related to the use of the resources, i.e. the number of times the user can access this service [14], periodicity-based, i.e. in which time intervals is this service going to be available, and even the version of the service, for example a trial version versus a full version. Separating service provisioning policies from authorization policies is especially useful when the authorizer and the service provider are two distinct yet collaborating entities. This approach achieves a fine grained control over the policies.
- 3) **Service Provider Privacy Policies:** This type of policy describes how a given service provider manages attributes and certificates of the entities it has interacted with. An example is represented by the current P3P policies [10] published at some web services. Service provider privacy policies have to meet the privacy preferences of the user intending to get the services.

ences of the user intending to get the services.

- 4) **Privacy Preferences Policies:** These policies specify the terms and conditions under which the attributes and/or certificates of the user can be shared within the federation. For example, a user can require that his/her attributes never be used for marketing. Here, we assume that attributes are only shared when a user attribute is required in order to complete a negotiation initiated by the user with the federation. Users should be able to express privacy preferences like “Share my attribute  $x$  with a service providers from which I have successfully received a service in the past.” Thus, while describing the user privacy preferences it should be possible to include the context of user’s past experience within the federation and other federation specific properties. For example there could be a reputation system in the federation; therefore a user could say which attributes should be shared with service providers according to the reputation of the SP.
- 5) **Federation Agreement Policies:** Federation agreement policies determine how to form a federation and what compliance rules and conditions a service provider needs to satisfy before it joins it. Additional agreements among a subset of service providers in the federation could also be specified through this type of policies.

An example to illustrate these different types of policy is given in what follows.

**Example 1** Consider a federation of three SPs. One is the Pharmacy which provides medicines to users. The second and third SPs (say HA and HB) collaborate together to provide the health information to users authorized to view this information. HA first authorizes the requester, and HB provides the information depending on how the user was authorized. HA is called the Health Information Authorizer, while HB is the Health Information Provider. The requester always interacts with HA in order to access the desired information. Also consider two users Alice and Nora representing two types of user. Alice is an external user and Nora is a nurse in the

*Pharmacy, and due to her affiliation with the Pharmacy she is also a user member of the federation.*

The main entities of the example are illustrated in Figure 1. Here, the different types of policy are represented using a legend format. First of all, federation agreement policies are enforced, to let the SPs to form the federation and establish internal collaborations. Once the federation is set, the Health Information Authorizer, shown in Figure 1, executes the following tasks: 1) authorizes the valid requesters, and 2) provides them information on how the user attributes will be used. Its activities are thus based on authorization policies and the privacy policies. The Health Information Provider, instead, 3) provides the service and therefore uses the service provisioning policies. Since the Pharmacy supports all three functions, it uses all the corresponding types of policy. Users Alice and Nora specify policies stating which entity is authorized to receive their attributes and certificates, and also under which conditions these can be shared in the federation. They therefore use the authorization policies and the user resources preferences policies.

Because enforcing all these types of policy require a large variety of information, it is important that the federated IdM systems be able to provide answers for various kinds of queries. This requires an expressive language not only to describe basic queries but also capture the history and the state of users' and providers activities. For example a service provider should be able to determine if a given user has successfully received some services by the federation. Also by analyzing the users' interactions, service providers would be able to detect malicious users trying denial of service type attacks.

The goal of our work is to present a policy language for federations, able to address all the mentioned requirements. The language we propose is based on a rich and expressive library of *assertions*. In the paper, we do not only present the grammar of the assertions, but we also define the underlying semantics of the proposed language constructs through a number of constraints defining assertion dependencies and implications. The constructs of our language can also be used for specifying protocols and for defining flexible and expressive policies. We provide an operational approach based on the assertions to support event logging, correctness checks of the user actions, and history tracking for the relevant events occurred in the federation. Finally, we also show how user information flow can be controlled and how we can infer user data without the need of ad hoc inference engine. To the best of our knowledge, this is the first time an assertion language supporting specific capabilities for managing identities of the user has been proposed.

The remaining of the paper is organized as follows. Next section presents an overview of our solution. Section III presents the library of assertions we have devised. In particular, in Section III-C we analyze the main constraints and logical relationships existing among the different assertions. In Section IV we present the grammar for policy formulation. In Section V we provide a relational representation of our

assertions and illustrate several advantages of our solution. Finally, in section VI we analyze work similar to ours and in Section VII we conclude the paper.

## II. OVERVIEW

Federations are characterized by different types of entity interacting among each other for various purposes. The entities involved in a federation are mainly users and SP's offering access to requesting users. Users are typically identified through SSO ids, assigned by IdP's. Interactions can occur between different types on entity: SP's and SP's for exchanging user data; users and SP's for accessing a resource, and provisioning of identity information; SP's and external authorities, for verifying users' data. Typically, a federation has an establishment phase, during which the different entities join, and an evolution phase, during which members interact among each other in order to exchange services and other information. Among all the possible interactions in a federation, in this paper we focus on actions related to authorization processes for member users. The process of authorization consists of granting an authenticated user access to a service he/she requested to a SP. We thus limit ourselves to considering actions that may be relevant for authorization decisions, such as access to a service.

In order to be authorized to services, users are required to satisfy specific conditions, stated in the corresponding policies. Here, we consider federations adopting attribute-based approaches to assign privileges to users<sup>1</sup>. As such our notion of *resource* includes both services, user attributes and certificates. Users might adopt privacy preference policies specifying whether or not attributes and certificates they own can be shared in the federation.

All the actions taken by SP's and users for authorization can be described as steps of pre-defined protocols. In our framework, we describe such actions using the notion of *assertion*. Each assertion is defined in terms of the main interacting actors, a time-stamp (if relevant for that operation), and other information relevant for authorization (such as the policy to be satisfied). Time plays a fundamental role in describing actions of a system. The validity and truth of some actions are actually strictly tied to the time the action is executed. We thus distinguish between assertions specifying static properties of the federation from time-based assertions. The first class of assertions, referred to as *specification assertions*, models actions representing properties that the referred entities possess. The second class of assertions, referred to as *execution assertions*, collects assertions that depend on specific actions of the entities in the federation. The assertions our language is composed of, capture the dynamic events occurring in the federation in a step by step, constructive approach. We are also able to specify authorization protocols by specifying the flow of actions that should be taken, and thus pose solid basis for the design of protocols able to detect possible misbehavior. Furthermore,

<sup>1</sup>Note that this approach is also valuable in case of role based approaches because roles can be described through attributes.

we express invariants in order to check security properties of the system and of the actions performed by the federation members. The policies we propose exploit the notion of assertions, so that a member can express any type of policy based on the previous actions recorded in the system. We also develop a relational representation of the federation log to keep track of the members activities. More precisely, from the architectural point of view, we consider a federation composed by multi-database systems, which include a database for each SP. Each database is under the control of the SP with which is associated and enables access to local data from the other SP's. Some middleware is used to allow a single query to span multiple SP's, without requiring the database servers to implement distributed query processing. This middleware is prevalent in current IdM systems and provides services such as identification, authentication, authorization, directories, and security. The log relation can be horizontally partitioned by the providers involved in the recorded events. Partitioning is needed since individual assertions are collected locally at each SP. These partitions can be combined when queries on the global log is made. The partitions of other SP's may also be replicated at a given SP if the records are referenced frequently for satisfying that SP's local policies. In the rest of the paper we however consider the classical view of multi-database system, which is viewed as a centralized database for its users. Figure 2 shows the architectural components of user clients and SP's. The *language and models* block in the center of the diagram corresponds to the main contributions of this paper, that is, the assertion library, the policy grammar, and the relational database with its respective access control related components. As shown, the most important components for a SP are: a module for policy manager, a subsystem collecting and managing remote attributes and credentials, and an access control module to enforce access control and policies. The user client components are: a local policy manager, an attribute and credential management system, and the interface by which the client interacts with the federation. In [15] a detailed SP architecture with the trust negotiation components was presented.

### III. LIBRARY OF ASSERTIONS FOR FEDERATION

In this section we formalize the building blocks for our language for federations. We start by defining the various symbols in the language. We then provide a comprehensive list of assertions modelling the events and the properties of a federation. We also illustrate the integrity constraints that should be preserved in order to ensure correctness of the system.

#### A. Federation Symbols

Symbols denote: *which* entities are involved in requesting and providing the services together with satisfying service requirements; *what* are the resources involved; and, *how* we determine that the access control on the resources is consistent with the original intentions of the resource owner. We define the variable symbols which are required in the specification

of the various policy constraints and assertions. Their values correspond to the respective types of constant symbols they represent.

**The Federation Constant Symbols.** The main elements in the federation framework are:

- *All users* ( $\mathcal{U}$ ) is the universal set of all possible users.
- *Users* ( $U \subseteq \mathcal{U}$ ) is the set of registered users who are identified by their single sign-on (SSO) ID. It is assumed that the users IdP can be inferred from the SSO of the user.
- *All service providers* ( $\mathcal{S}$ ) is the universal set of all possible service providers.
- *Service Providers* ( $S \subseteq \mathcal{S}$ ) is a set of service providers within the federation under consideration.<sup>2</sup> SP's are identified by unique identifiers such as, for instance, their associated public keys.
- *All external trusted third parties* ( $\mathcal{E}$ ) is the universal set of all parties outside the federation, which are referred for the verification of certificates issued by them.
- *Resources* ( $R$ ) denotes the set of resources available in the federation. Resources are classified into three possible classes, that is, attributes, certificates and services. Each class has an associated set of descriptors, collecting attribute types (denoted as  $\alpha$ ), certificate types (denoted as  $\phi$ ), and service types ( $\psi$ ), respectively. Each resource  $r \in R$  is always defined in the context of  $U$  or  $S$ . For example, we denote attribute of type  $\alpha_1$  of user  $U_1$  as  $R_{U_1}^{\alpha_1}$ . Similarly, we denote resource of type  $\phi_1$  of  $SP'$  as  $R_{SP'}^{\phi_1}$ . If the owner of the resource or the type of resource is unknown or not relevant in a given context, then a special symbol  $\perp$  is used instead of the respective symbol. For example, when the attribute type of a user  $U_i$  is not relevant we use  $R_{U_i}^{\perp}$ .
- *Policy constraints or Policies* ( $\mathcal{C}$ ) is a set of policies used to establish authentication and authorization requirements for elements in  $R$ , and to regulate sharing of resources in the federation. We assume that for each  $R_{\perp}^{\perp}$  there is a policy set  $\mathcal{C}_{R_{\perp}^{\perp}} \subseteq \mathcal{C}$  defined by the corresponding resource owner which does not change over time. The subset can possibly be null for unprotected resources. Different types of policies are expressed in  $\mathcal{C}$ . According to the taxonomy proposed earlier in the paper,  $\mathcal{C}$  contains *Resource Authorization Policies* for each SP resource  $r_s^{\psi_1}$ .  $\mathcal{C}$  will contains the policies specified by the users, that is, policies for protecting attributes and certificates, classified as *Privacy Preferences*. Set  $\mathcal{C}$  might also contain SP's organizational policies for preserving the privacy of its users, called *Service Provider Privacy policies*. We do not include *Federation Agreement Policies* in the  $\mathcal{C}$  set, since they are usually off-line business agreements like [7] and their formalization is outside the scope of this paper.
- *Time* ( $\mathbb{T}$ ) is the discrete time in the system. We assume

<sup>2</sup>In this paper we consider one federation, thus multiple federations are outside the scope of this paper.

closely synchronized clocks in the federation to uniformly record the time.

- *Natural Numbers* ( $\mathbb{N}$ ).

**The Federation Variable Symbols.** Variable symbols corresponding to the constant symbols given are denoted by  $V_U, V_S, V_E, V_U, V_S, V_{R^\perp}, V_C, V_T$  and  $V_{\mathbb{N}}$  respectively. The variable symbols are placeholders in policy statements for their respective types of constant symbols. All user terms denoted by  $UT$  would therefore be equal to the set  $V_U \cup U \cup V_U \cup \mathcal{U}$ . Similarly we have service provider terms  $ST$ , external authority terms  $\mathcal{ET}$ , resource terms  $RT$  containing  $R^\alpha_\perp T, R^\phi_\perp T, R^\psi_\perp T$ , policy terms  $CT$ , terms denoting time as  $\mathbb{T}$  and finally the number terms  $\mathbb{N}$ .

**The Federation Assertion Symbols.** The assertions we provide are organized into four distinct sets: (1) a set of specification assertions ( $\mathcal{SP}$ ) which provide the facts and assertions related to the entities in the federation; (2) a set of execution assertions ( $\mathcal{EP}$ ) which help in determining the events; (3) a set of planning assertions ( $\mathcal{PP}$ ) which give the restrictions to disallow or mandate  $EP \subseteq \mathcal{EP}$ ; and (4) a set of comparison assertions ( $\mathcal{CP}$ ), to represent the comparison operations. Comparison operations include the binary predicate  $=$ , the arguments of which are elements in  $UT, ST, RT, CT$  and  $\mathbb{N}$ , and binary predicates  $<, \leq, >, \geq$ , the arguments of which are elements in  $\mathbb{N}$  and  $\mathbb{T}$ . Assertions in  $\mathcal{SP}, \mathcal{EP}$  and  $\mathcal{PP}$  are described in Tables I, II and III, respectively. The arguments of the assertions are typically simple terms from  $UT, ST, RT, CT$  and  $\mathbb{N}$ . More complex assertions can be specified by combining other assertions as given in Table III for  $\mathcal{PP}$ . Here, a conjunction of  $\mathcal{SP}, \mathcal{EP}, \mathcal{CP}$  and  $\mathcal{PP}$  itself is used. The semantics of the language and detailed explanation of the most significant assertions are given in the next section.

## B. Semantics of Assertions

In this section we provide the semantics of the introduced assertions and specify the relationship among them. In particular, in Section III-C we discuss the main implications among the most interesting assertions.

**1. Specification assertions ( $\mathcal{SP}$ ).** Specification assertions can be determined statically for a given federation. By statically we mean that no specific action has to be executed to determine their truth values. As such the validity of this type of assertion can always be checked, and therefore they do not depend on time. Non-obvious details of some of the assertions belonging this class are as follows.

- 1) *possess\_resource*( $x, r_x^\perp$ ): A user or SP may *possess* a resource even if it does not own it. This assertion models the case of a particular SP acting as the IdP for a particular user and providing the identity attributes of that user. This assertion helps in identifying the location of resource  $r_x^\perp$ .
- 2) *resource\_authr\_policy*( $x, r_x^\perp, C_{r_x^\perp}$ ): If this assertion is true for  $x$ , which is either a user or a service provider,  $x$  can provide resource  $r_x^\perp$  if the policy  $C_{r_x^\perp}$  is satisfied.
- 3) *SP\_authn\_policy*( $sp, C_\perp$ ): This assertion states that the  $C_\perp$  policy requirements of  $sp$  were satisfied when the

user was authenticated. For a subject to be seamlessly authenticated from one SP to another it should be possible for an SP to determine how the subject was authenticated in the first place.

**2. Execution assertions ( $\mathcal{EP}$ ).** Execution assertions model dynamic *events*. The truth of such assertions is given by the actions of a user or a SP at a given point in time. Because the time of each event is recorded, execution assertions are the key to determine the history of events in the federation. The complete set of assertions belonging this class is reported in Table II. We describe the most significant ones in more detail in what follows.

- 1) *resource\_request*( $x_{req}, x_{prov}, r_{x_{prov}}^\perp$ ): The semantics of this assertion is that  $x_{req}$  has requested  $x_{prov}$  for resource  $r_{x_{prov}}^\perp$ .  $x_{req}$  and  $x_{prov}$  could be either a SP or a user. The resource  $r_{x_{prov}}^\perp$  could be an attribute  $r_{x_{prov}}^\alpha$ , certificate  $r_{x_{prov}}^\phi$  or service  $r_{x_{prov}}^\psi$ .
- 2) *authorize\_access*( $r_{req}, C_{r_{prov}}^\perp, t$ ): If this assertion is true, then the requester has satisfied all the conditions given in the policy constraints and is qualified to receive the resource  $r_{prov}^\perp$  at time  $t$ .  $r_{req}$  here represents the main elements of a *resource\_request*, that is: the requester, which is either a user or a SP; the provider, which again is either a user or a SP; and a policy. Authorization decisions are made through a negotiation process; basically such process requires verifying that the attributes provided by the requester satisfy conditions of relevant authorization policies.
- 3) *provide\_resource*( $r_{req}, t$ ): The truth of this assertion depends on the type of resource in  $r_{req}$ . If the resource is some identity information, that is,  $r_{x_{prov}}^\alpha$  or  $r_{x_{prov}}^\phi$ , then this assertion implicitly provides information on which of the user's identity information is released to the provider. In case the resource is a service  $r_{x_{prov}}^\psi$ , the assertion indicates the time from when the resource is available for access.
- 4) *verify\_resource*( $x, r_{owner}^\perp, t$ ): The truth value of this assertion essentially states the validity of the information provided by the resource. Typically,  $r_{owner}^\perp$  is either a  $r_{owner}^\alpha$  or  $r_{owner}^\phi$ . The issuer of the certificate is usually the designated verifier, and it can also be outside the federation.
- 5) *begin\_access*( $x_{req}, x_{prov}, r_{prov}^\perp, t$ ): If this assertion is true, the requester  $x_{req}$  has started accessing the resource  $r_{prov}^\perp$  at time  $t$ . The time  $t$  is relevant especially for cases in which the access time is different from the time the corresponding authorization (see assertion 2) was granted.
- 6) *abort\_access*( $x_{req}, x_{prov}, r_{prov}^\perp, t$ ): If this assertion is true, the requester  $x_{req}$  started accessing the resource  $r_{prov}^\perp$  but aborted the access at time  $t$ . The information about abortion of a resource access may be used in a policy as criteria for providing (or denying) services.
- 7) *success\_access*( $x_{req}, x_{prov}, r_{prov}^\perp, t$ ): If this assertion is true, the requester  $x_{req}$  successfully completed accessing

Assertion	Arity	Argument Types	Meaning
<i>registered</i>	1	$UT$	If $registered(u_i)$ is true it would mean that $u_i \in U$ and is represented by a valid SSO identifier .
<i>member</i>	1	$UT$	If $member(u_i)$ is true then the user $u_i \in U$ is affiliated or is an employee of an organization in the federation.
<i>SP_member</i>	1	$ST$	If $SP\_member(s_i)$ is true then the service provider $s_i \in S$ and is a valid federation SP.
<i>resource_owner</i>	2	$UT ST, RT$	If $resource\_owner(x, r_x^\perp)$ is true then $x \in (U \cup S)$ owns resource $r_x^\perp \in R$ which can be provided under policy restrictions.
<i>resource_possessor</i>	2	$UT ST, RT$	If $resource\_possessor(x, r_x^\perp)$ is true then $x \in (U \cup S)$ has resource $r_x^\perp \in R$ which it does not own but which can be provided under policy restrictions.
<i>resource_verifier</i>	2	$ST \mathcal{E}T, RT$	If $resource\_verifier(x, r_x^\perp)$ is true then $x \in (\mathcal{E} \cup S)$ is authorized to verify the validity of the resource $r_x^\perp \in R$ . Generally the resource verifier is the actual issuer of the resource but there could be delegated authorities within and outside the federation which can do the verification
<i>resource_authr_policy</i>	3	$UT ST, RT, CT$	If $resource\_authr\_policy(x, r_x^\perp, C_{r_x^\perp})$ is true then $x \in (U \cup S)$ offers resource $r_x^\perp \in R$ and has authorization policies $C_x \in \mathcal{C}$ for that resource.
<i>SP_authn_policy</i>	2	$ST, CT$	If $SP\_authn\_policy(sp, C_\perp)$ is true then $sp \in S$ has authentication policies $C_\perp$ . These authentication policies specify which user attributes are verified to authenticate this user.

TABLE I  
SPECIFICATION ASSERTIONS

Assertion	Arity	Argument Types	Meaning
<i>authn_user</i>	4	$UT, ST, CT, \mathbb{T}T$	If $authn\_user(u, sp, C_\perp, t)$ is true then at time $t$ , user $u$ was successfully authenticated by SP $sp$ in accordance to the authentication policies $C_\perp$ .
<i>resource_request</i>	4	$(UT ST), (UT ST \mathcal{E}T), RT, \mathbb{T}T$	If $resource\_request(requester, provider, r_{prov}^\perp, t)$ is true, then at time $t$ , the requester requested for the provider's resource $r_{prov}^\perp$ .
<i>authorize_access</i>	5	$rreq^3, CT, \mathbb{T}T$	If $authorize\_access(rreq, C_{r_{prov}^\perp}, t)$ is true then the requester successfully qualified for the provider's resource at time $t$ by satisfying the policy constraints $C_{r_{prov}^\perp}$ .
<i>provide_resource</i>	4	$rreq, \mathbb{T}T$	If $provide\_resource(rreq, t)$ is true, then at time $t$ , provider is enabled access or transmitted the requested resource to the requester.
<i>verify_resource</i>	3	$(ST \mathcal{E}T), RT, \mathbb{T}T$	If $verify\_resource(x, r_{owner}^\perp, t)$ is true then at time $t$ , $x \in S \cup \mathcal{E}$ verified the validity of the resource $r_{owner}^\perp$ .
<i>begin_access</i>	4	$rreq, \mathbb{T}T$	If $begin\_access(rreq, t)$ is true then at time $t$ , the requester who was qualified for the resource has started accessing that resource.
<i>abort_access</i>	4	$rreq, \mathbb{T}T$	If $abort\_access(rreq, t)$ is true then the requester qualified for the resource but still aborted the transaction at time $t$ .
<i>success_access</i>	4	$rreq, \mathbb{T}T$	If $success\_access(rreq, t)$ is true then the requester who was qualified for the resource got it successfully at time $t$ .

TABLE II  
EXECUTION ASSERTIONS

Assertion	Arity	Argument Types	Meaning
<i>cannot_access</i>	3	$UT ST$ , conjunction of specification, execution or comparison and atomic literals, $R$	If $cannot\_access(x, \{registered(..) \wedge authorize\_access(..)\}, R)$ is true it would mean that if requester $x$ is $registered(..)$ and has successfully qualified for the access the resource defined in that assertion, then resource $R$ cannot be granted to $x$ .
<i>must_access</i>	3	$UT ST$ , conjunction of specification execution or comparison and atomic literals, $R$	If $must\_access(\{x, success(..medicine)\}, resource\_SideAffects)$ is true it would mean that if requester $x$ has successfully got a <i>medicine</i> then that entity should also access <i>resource_SideAffects</i> .

TABLE III  
PLANNING ASSERTIONS

the resource  $r_{prov}^\perp$  at time  $t$ .

**3. Planning assertions ( $\mathcal{PP}$ ).** We define two planning assertions. The first one specifies the *restrictions* for a resource, while the other specifies *obligations* if certain constraints are satisfied. These assertions are related to a resource and defined by the owner or possessor of that resource. Examples illustrating the usage of these assertions are provided in Figure 4.

- 1) *cannot\_access*( $x_0, \wedge\{\text{conjunction of } \mathcal{SP}, \mathcal{EP}, \mathcal{CP}, \mathcal{PP} \text{ and/or terms}\}, r_{prov}^\perp$ ). This assertion states the prohibition for a user  $x_0$  to access resource  $r_{prov}^\perp$ . It is defined using the terms, introduced in Section III-A, and assertions themselves.  $x_0$  (either a user or a service provider) verifies this assertion, if it verifies the conditions given by the conjunction of assertion instances and atomic terms.
- 2) *must\_access*( $x_0, \wedge\{\text{conjunction of } \mathcal{SP}, \mathcal{EP}, \mathcal{CP}, \mathcal{PP} \text{ and/or terms}\}, r_{prov}^\perp$ ). If this assertion is true then the entity  $x_0$ , either a user or a service provider, satisfies the conditions given by the conjunction of assertion instances and atomic terms, and is therefore obligated to access the resource  $r_{prov}^\perp$ .

### C. Derivation rules for Assertions

Derivation rules, also referred to as *implications*, are used to represent logical implications holding among different assertions and integrity constraints. Satisfaction of such constraints should be verified as the corresponding actions occur. The number of possible implications is high, and we report here the most interesting ones.

- *resource\_possessor*( $x, r_x^\perp$ )  $\rightarrow$  *registered*( $x$ )  
This implication is time-invariant; it states that if an entity possesses a resource, then it is a registered entity in the federation.
- *resource\_owner*( $x, r_x^\perp$ )  $\rightarrow$  *resource\_possessor*( $x, r_x^\perp$ )  
This implication is time-invariant; it states that if an entity owns a resource, then it also possesses it.
- *resource\_authr\_policy*( $x, r_x^\perp, C_{r_x^\perp}$ )  $\rightarrow$   $\{x \in U \cup S, r_x^\perp \in R, C_{r_x^\perp} \in \mathcal{C} | \text{resource\_possessor}(x, r_x^\perp)\}$   
This implication is time-invariant; it states that if an entity has authorization policies for a resource, then it also manages that resource.
- *resource\_request*( $x_{req}, x_{prov}, r_{x_{prov}}^\perp, t_1$ )  $\rightarrow$   $\{x \in U \cup S, r_x^\perp \in R | \text{possess\_resource}(x_{prov}, r_{x_{prov}}^\perp)\}$   
This implication states that a resource request is valid only if the provider also possesses the resource.
- *authorize\_access*( $x_{req}, x_{prov}, r_{x_{prov}}^\perp, C_{r_{x_{prov}}^\perp}, t$ )  $\rightarrow$   $\{x_{req} \in U \cup S, x_{prov} \in U \cup S, r_{x_{prov}}^\perp \in R, C_{r_{x_{prov}}^\perp} \in \mathcal{C}, t \in \mathbb{T} | \text{resource\_request}(x_{req}, x_{prov}, r_{x_{prov}}^\perp, t_1) \wedge (t_1 < t) \wedge \text{resource\_authr\_policy}(x_{prov}, r_{x_{prov}}^\perp, C_{r_{x_{prov}}^\perp})\}$   
This implication depends on the time the *authorize* event occurred. If an entity (the provider) authorizes a requester for a resource, this implies that at an earlier time the requester had requested such resource. It also implies

that the provider possesses the resource and the related authorization policies  $C_{r_{x_{prov}}^\perp}$  are satisfied.

- *begin\_access*( $x_{req}, x_{prov}, r_{x_{prov}}^\perp, t_2$ )  $\rightarrow$   $\{x_{req} \in U \cup S, x_{prov} \in U \cup S, r_{x_{prov}}^\perp \in R, t_2 \in \mathbb{T} | \text{authorize\_access}(x_{req}, x_{prov}, r_{x_{prov}}^\perp, t_1) \wedge (t_1 < t_2)\}$   
This implication requires the beginning of an access to be subsequent to a corresponding request successfully authorized by the service provider.
- *abort\_access*( $x_{req}, x_{prov}, r_{x_{prov}}^\perp, t_2$ )  $\rightarrow$   $\{x_{req} \in U \cup S, x_{prov} \in U \cup S, r_{x_{prov}}^\perp \in R, t_1 \in \mathbb{T} | \text{begin\_access}(x_{req}, x_{prov}, r_{x_{prov}}^\perp, t_1) \wedge (t_1 < t_2)\}$
- *success\_access*( $x_{req}, x_{prov}, r_{x_{prov}}^\perp, t_3$ )  $\rightarrow$   $\{x_{req} \in U \cup S, x_{prov} \in U \cup S, r_{x_{prov}}^\perp \in R, t_1 \in \mathbb{T} | \text{begin\_access}(x_{req}, x_{prov}, r_{x_{prov}}^\perp, t_1) \wedge (\text{not}(\text{abort\_access}(x_{req}, x_{prov}, r_{x_{prov}}^\perp, t_2))) \text{ for } t_0 \leq t_1 \leq t_2 \leq t_3\}$   
The two last implications state that either the abort or the success of an access operation is subsequent to a corresponding begin access.

Based on these derivation rules, we now introduce the notion of *terminal assertions*. Terminal assertions essentially mark the end of a protocol, such as *success\_access*, *abort\_access*. If a terminal assertion is true, then all the other assertions it implies are also true. The terminal assertions are inferred by using the above derivation rules presented. When the terminal assertion is the head<sup>4</sup> of a derivation rule, the assertions given in the body of the rule is recursively unfolded to determine the transitive closure of this terminal assertion. For example *success\_access* is the terminal assertion with *begin\_access* in its body. Also, *begin\_access* is the head for *authorize\_access* which in turn is the head for *resource\_request*. Therefore the closure of *success\_access* is  $\{\text{begin\_access} \triangleleft \text{authorize\_access} \triangleleft \text{resource\_request}\}$  where  $\triangleleft$  denotes an ordering in the closure. The terminal assertion can essentially assert the truth value of its closure in the defined order. Regarding the specification assertions, the terminal assertion is *resource\_authr\_policy*. For the execution assertions the defined terminal assertions are *success\_access* and *abort\_access*.

Conflicting requirements can be detected automatically if the set of conflicting assertions are pre-defined by the system. Objects for which this type of requirements can be checked are either policies, as specified in Section IV, or logs. (see Section V). Some interesting conflicting assertions, denoted by operator  $\otimes$ , are listed below: *abort\_access*( $r_{req}, t$ )  $\otimes$  (*success\_access*( $r_{req}, t_1$ )) where  $t_1 \geq t$   
*cannot\_access*( $x, R$ )  $\otimes$  *must\_access*( $y, R$ )  
*cannot\_access*( $x, R$ )  $\otimes$  *authorize\_access*( $r_{req}, y$ )  
*must\_access*( $x, R$ )  $\otimes$  *cannot\_access*( $y, R$ )  
*must\_access*( $x, R$ )  $\otimes$  *authorize\_access*( $r_{req}, y$ )

<sup>4</sup>If  $A \rightarrow B \wedge C$  where  $A$ ,  $B$  and  $C$  are assertions,  $A$  is called the head and  $B$  and  $C$  is called the body.

```

policy_set      ::= targets provider targets
::= target targets
               | target
target         ::= policy_conditions requester provider
resource
               % target denotes under which conditions
               % given resource 1) can 2) cannot and
               % 3) must not be accessed
requester      ::= service_provider_id      % SP member if the federation identifier
               | user_id                  % User SSO id in the federation
               | _                        % wildcard when the exact id of requester
               % is not relevant
provider       ::= service_provider_id
               | user_id
               | _
resource       ::= resource_id              % resource id in the federation
policy_conditions ::= true                  % empty policy
               | false                    % policy can never be satisfied
               | one_policy_condition (and) policy_conditions % conjunction of policies
               | one_policy_condition (or) policy_conditions  % disjunction of policies
               | one_policy_condition      % one single policy
one_policy_condition ::= specification_assertion % static assertion of a federation
               | execution_assertion        % dynamic event in a federation
               | planning_assertion        % access permission, obligation and
               % restriction for a possibly different resource
               | (not) specification_assertion
               | (not) execution_assertion
               | attribute comparison_assertion number % Eg. Age < 25
               | certificate
               | attribute

```

Fig. 3. Policy BNF Grammar

#### IV. POLICY FORMULATION

Every user and service provider in the federation has a set of resources to share. Generally speaking, the policy for any resource  $r$  defines the conditions which need to be satisfied before the resource can be released to the requester. As mentioned, several types of policy can be used in a federation for different purposes. In this section we focus on the most relevant policy types that are needed in a federation, that is, the *resource authorization policies* and the *user privacy preferences policies*. For the formulation of both kinds of policies, a grammar is defined (see Figure 3). We use the Backus-Naur notation to describe the syntax of the policy language. The proposed grammar is composed of eight main rules. The starting symbol is `policy_set` which, as the name implies, is the set of policies for a federation entity ( $U$  or  $S$ ). The owner of this resource is referred as `provider`. The `policy_set` is basically one or more `targets`. Each `target` is `policy_conditions` associated with a given resource, `requester` and `provider`. If a generic policy for a resource has to be specified, then wildcard value `_` is used instead of the `requester` or `provider`. The `target` is always specified in the context of a resource. `policy_conditions` symbol, in turn, consists of the conjunction or disjunction of one or more conditions expressed in terms of federation variables and assertions.

To illustrate how the grammar can be used to express the different kinds of policies, we elaborate on Example 1 and show the corresponding policy encoding in Figure 4. In the example, we report only the policies relevant to the example scenario. The entities could have several additional policies in addition to the ones highlighted.

**Example 2** *The on-line pharmacy ( $SP_{Phar}$ ) sells two drugs  $M_1$  and  $M_2$ .  $SP_{Phar}$  also possesses reports providing information about the corresponding side effects the drugs, namely  $MS_1$  and  $MS_2$ , which must be disclosed to the drug buyer when the respective drug is released. It is not safe to release  $M_1$  and  $M_2$  to the same subject. Therefore, the pharmacy should check that if either drug was acquired before by a user, he/she should not be allowed to obtain the second drug. In addition to the above restriction, for users that are not members of the federation, the pharmacy needs to determine the name of the user, a verification that his/her age being greater than 25 and a valid credit card number. For users who are members of the federation, there is a special deal where only the age restriction is required. A specification of such policy in our language is given in Figure 4, part (a).*

*Consider now the scenario of two users Alice and Nora. Alice is a registered yet non-member user of the federation. She does not have any authorization policy for her name and age attributes but she wants to make sure that the other party has a BBB certificate before releasing her credit card number. She is willing to federate<sup>5</sup> her name and age but not her credit card number. The corresponding policies are reported in parts (b) and (c) of Figure 4.*

*Nora, on the other hand, is the nurse who because of her profession is a member of the federation. Her age can only be disclosed to members of the federation. Also, because for members we assume their federated attributes are available with their affiliated service provider, Nora does not federate her attribute. Nora's policy is shown in parts (d) and (e).*

<sup>5</sup>An attribute or certificate which is federated means that it can be shared in the federation i.e. given to other service providers in the federation when required by them.

(a) Service Provider : Pharmacy's Resource Authorization Policy Set.

1.  $AuthPol\_SP_{Pharm} := must\_access(success\_access(V_U, SP_{Pharm}, M1), MS1)$
2.  $must\_access(success\_access(V_U, SP_{Pharm}, M2), MS2)$
3.  $cannot\_access(success\_access(V_U, SP_{Pharm}, M1), M2)$
4.  $cannot\_access(success\_access(V_U, SP_{Pharm}, M2), M1)$
5.  $resource\_authr\_policy(V_U, SP_{Pharm}, R_{Pharm}, C_{Med})$
6.  $C_{Med} := registered(V_U) \text{ (and)}$
7.  $((member(V_U) \text{ (and)}$
8.  $(age_{V_U} > 25)) \text{ (or)}$   
 $(name_{V_U} \text{ (and)} (age_{V_U} > 25) \text{ (and)} CreditCard_{V_U}))$

Here resources  $R_{Pharm}$  of the pharmacy include the medicines  $M1$  and  $M2$  and their corresponding prescriptions  $MS1$  and  $MS2$ .

(b) Non Member User: Alice's Disclosure Policy.

9.  $AuthPol\_Alice := resource\_authr\_policy(V_S, Alice, name, (SP\_member(V_S)))$
10.  $resource\_authr\_policy(V_S, Alice, age, (SP\_member(V_S)))$
11.  $resource\_authr\_policy(V_S, Alice, CreditCard, (SP\_member(V_S) \text{ (and)}$   
 $BBB_{V_S}))$

(c) Non Member User: Alice's Privacy Preferences Policy.

12.  $P^2Pol\_Alice := resource\_authr\_policy(V_S, Alice, name, (SP\_member(V_S)))$
13.  $resource\_authr\_policy(V_S, Alice, age, (SP\_member(V_S)))$
14.  $resource\_authr\_policy(V_S, Alice, CreditCard, false)$

(d) Member User: Nora's Disclosure Policy.

15.  $AuthPol\_Nora := resource\_authr\_policy(V_S, Nora, age, (SP\_member(V_S)))$

(e) Member User: Nora's Privacy Preferences Policy.

12.  $P^2Pol\_Nora := resource\_authr\_policy(V_S, Nora, age, false)$

Fig. 4. Example Policy Sets

## V. RELATIONAL MODEL, QUERY TYPES AND IDENTITY INFORMATION FLOW

In this section we provide an operational approach to support the assertion language we have developed. We propose to use a log of the actions executed by the entities in the federation; such log is stored at a trusted site. The log can be implemented through a dedicated server which collects and stores messages corresponding to execution assertions reporting actions by the interacting entities in the federation. The log relation can be horizontally partitioned by the providers involved in the recorded events.<sup>6</sup> In our approach the log is a relational table, defined according to the notion of relation of the relational data model. Checks for the log consistency are encoded using SQL-like queries. To ease the presentation we use the log table given in Table IV called ASSERT\_LOG as example.

As shown, log records are sorted by time, and each entry keeps track of the assertion type and arguments. Time symbol  $T_0$  is used to denote time-invariant specification assertions. Each time an assertion is added to the ASSERT\_LOG table, its validity is to be verified. According to our relational model, the translation of the consistency checks illustrated in Section III-C is executed by some queries on the ASSERT\_LOG table. As an illustrative example, consider the insertion of

$authorize\_access$  at line 8. The following PL/SQL function translates the consistency check to be executed when this type of assertion is to be inserted.

```
CREATE OR REPLACE FUNCTION
check_insert_authr_access(
  t ASSERT_LOG.Time%Type,
  req ASSERT_LOG.Requester%Type,
  prov ASSERT_LOG.Provider%Type,
  resource ASSERT_LOG.Resource%Type,
  pol ASSERT_LOG.Policy%Type)
RETURN BOOLEAN IS
BEGIN
  -- check for resource_request
  SELECT *
  FROM ASSERT_LOG AL
  WHERE (AL.Assertion = "resource_request") AND
        (AL.Requester = req) AND
        (AL.provider = prov) AND
        (AL.Resource = resource) AND
        (AL.Time < t)
  AS Cond1;
  -- check for resource_authr_policy
  SELECT *
  FROM ASSERT_LOG AL
  WHERE (AL.Assertion =
        "resource_authr_policy") AND
        (AL.Requester = req) AND
        (AL.provider = prov) AND
```

<sup>6</sup>Due to lack of space we do not further elaborate on the federation architecture in this paper.

	Time	Assertion	Requester	Provider	Resource	Policy
1	T0	<i>registered</i>	Alice@SP1			
2	T0	<i>resource_owner</i>		SP2	online-book	
3	T0	<i>resource_authr_policy</i>		SP2	online-book	SP2-BookPol*
4	T1	<i>resource_request</i>	Alice@SP1	SP2	online-book	
5	T2	<i>resource_request</i>	SP2	Alice@SP1	$CCN_{Alice}$	
6	T3	<i>provide_resource</i>	SP2	Alice@SP1	$CCN_{Alice}$	
7	T4	<i>verify_resource</i>	SP2	CCN-Authority	$CCN_{Alice}$	
8	T5	<i>authorize_access</i>	Alice@SP1	SP2	online-book	SP2-BookPol*
9	T6	<i>provide_resource</i>	Alice@SP1	SP2	online-book	
10	T7	<i>begin_access</i>	Alice@SP1	SP2	online-book	
11	T8	<i>success_access</i>	Alice@SP1	SP2	online-book	

TABLE IV  
EXAMPLE OF ASSERT\_LOG.

\*SP2-BookPol:=  $CCN_{req} \wedge count(abort\_access(req, SP2, online-book, t))$  where  $curTime - T < 10min$

```

(AL.Resource = resource) AND
(AL.Policy < pol)
AS Cond2;
IF ((Cond1 != NULL) and
(Cond2 != NULL)) THEN
RETURN true;
ELSE
RETURN false;
END IF END check_insert_authr_access;

```

The ASSERT\_LOG can also be exploited to evaluate aggregate queries concerning the system, such as, "How many resource requests have been made to SP2?". Such query requires to check the *resource\_request* assertion in the log such that the provider is SP2. It can be formulated as follows:

```

SELECT COUNT(*) FROM ASSERT_LOG AL WHERE
AL.PROVIDER='SP2' AND
AL.Assert = "resource_request"

```

Aggregate queries are useful in understanding the state or behavior of the entities. The result of these queries may be required to verify the satisfaction of policy conditions or even analyze the security of the system by detecting anomalies. An example of a policy condition is "If a user has aborted more than 10 transactions then disallow the current resource request". Regarding anomaly detection using assertions, a possible application is the following. Consider a SP that typically manages an average of 100 requests per day. If the result of the aggregate query given above returns a value very different from 100 this would result in the notification of an anomaly.

Temporal queries are another type of query which is important for auditing and checking activities in the federation. Advances in temporal query languages [13], [3] show that several applications may benefit substantially from the capability of issuing temporal queries. To answer these types of query in our system, the *Time* column of the ASSERT\_LOG has to be considered. For example, if the auditing mechanism needs to detect whether an *abort* occurred the following query can be issued: "If any service access began at time  $T$  and has not been completed successfully even by a defined time say  $T + SysDef.WaitTime$ , then it is a possible abort." The

query that asks to retrieve possible aborts would be as follows:

```

SELECT P.Time as t, P.Requester, P.Provider,
P.Resource
FROM ASSERT_LOG P WHERE
(SELECT * FROM ASSERT_LOG P1
WHERE (P1.Assertion = "success_access") AND
(P1.Requester = P.Requester) AND
(P1.Provider = P.Provider) AND
(P.Resource = P.Resource) AND
(P.Time <= t + SysDef.WaitTime))
IS NULL

```

This example shows how a SP can detect *timeouts* by executing the above query periodically. If the SP can detect possible *aborts*, then it does not waste resources keeping the state of an open request. This increases robustness of the SP's system and prevents security threats like denial of service.

Updating the ASSERT\_LOG by using this detection mechanism also helps in keeping the log in a consistent state. Future inferences from the log can then be made correctly. Updating also allows efficient and correct retrieval of the history of activities in the system.

Our solution also provides mechanisms supporting a compact representation of the ASSERT\_LOG based on the semantics of the assertions. In order to remove entries in the log file without losing important information, we need to identify the minimal set of assertions sufficient to reconstruct the history. The notion of terminal assertions introduced in Section III-C can be used to this extent. For example, static and time-dependent terminal assertions are at line 3 and 11 of Table IV, respectively. By removing the assertions belonging to the closure of each of these assertions, we obtain Table V. The information loss during the reduction process is related with the temporal references associated with the removed assertions. Therefore, the compact representation of ASSERT\_LOG may not be adequate for answering precise temporal queries. However, for all other types of query, the compacted assertions can be retrieved by a recursive unfolding using the derivation rules.

Another important feature achieved by the usage of

	Time	Assertion	Requester	Provider	Resource	Policy
2	T0	<i>resource_owner</i>		SP2	online-book	
3	T0	<i>resource_authr_policy</i>		SP2	online-book	SP2-BookPol*
11	T8	<i>success_access</i>	Alice@SP1	SP2	online-book	

TABLE V  
EXAMPLE OF COMPACTED ASSERTION LOG.

\*SP2-BookPol:=  $CCN_{req} \wedge count(abort\_access(req, SP2, online-book, t))$  where  $curTime - T < 10min$

ASSERT\_LOG is the possibility of reasoning about the **flow of identity information** of the users. In any IdM controlling how the identity information of its users is retrieved and shared is vital. Preventing improper propagation of such information is crucial to the security of the federation system. Flow of identity information happens when a user's attribute or certificate is transferred from one entity to another. This requires enforcement of fine-grained access control policies among SP's, to prevent unauthorized identity information leaks. Information about the flow of users identity information can be used when enforcing privacy control, compliance checking and detecting anomalous sharing of identity information. Operatively, information about user's identity information can be retrieved from the ASSERT\_LOG by identifying the *provide\_resource* assertions. The record retrieved by selecting a *provide\_resource* assertion clearly states where the resource (which in this case is either a certificate or an attribute) has been transferred. The assertion should satisfy one of the following two conditions:

- 1) The provider should be a valid SSO ID identifying a user releasing its attribute or certificate. In the reported ASSERT\_LOG, this case is shown at line 6.
- 2) The resource is a user attribute of type  $\alpha_i$ , or a certificate of type  $\phi_i$ , that is,  $r_U^{\phi_i}$ . The actual name of the resource can be retrieved from the Resource field of the ASSERT\_LOG.

Identity information can be leaked even when the actual value of the identity attribute or certificate is not shared. Identity information can in fact be inferred by analyzing the flow of certain assertions. For example, at line 3 the policy for resource online-book is SP2-BookPol\*, requiring a  $CCN_{req}$ . At line 8 the assertion *authorize\_access(Alice@SP1, SP2, online-book, SP2 - BookPol\*, T5)* is stored, showing that Alice satisfied the online-book policy. As a result, it can be inferred that  $CCN_{Alice}$  is given to SP2. Note that inference is possible even if for privacy or security reasons, only a restricted view of the entire table is available, and not all the entries are shown.

## VI. RELATED WORK

In this section we discuss related work. In particular, we focus on the assertion based language SAML [8], because it has similar goal to our work. We then briefly overview some significant policy languages, such as Rei [11] and Ponder [4] and compare them with our work.

The Security Assertion Markup Language (SAML) [8] is an XML standard for exchanging authentication and authorization data between security domains, that is, between an identity provider and a service provider. SAML has been developed by the OASIS Security Services Technical Committee. SAML supports the specification of assertions about a subject that can be shared with other applications across system domain boundaries. It allows an entity to make assertions regarding the identity, attributes, and entitlements of a subject or a user to other entities. In a federation system, these assertions are usually transferred from IdP's to SP's. Assertions contain statements that SP's use to make access control decisions. There are three main statements provided by SAML: 1) *Authentication statements*: Such a statement asserts to the SP that the principal did indeed authenticate with the IdP at a particular time using a particular method of authentication. 2) *Attribute statements*: Such a statement asserts that the specified subject is associated with the supplied attributes. 3) *Authorization decision statements*: Such a statement specifies that a given subject has been allowed or denied access to a given resource. SAML is extensively used for authentication and authorization information exchange by most of the federated identity management initiatives [6], [9].

A major difference of our approach with respect to SAML is that our language does not only supports the specification of authentication and authorization information, but it also supports the specification of the static and dynamic properties of a federation. Such a capability is important in order to perform security analysis and to capture the history of events in the system. The various assertions provided as part of our assertion library also represent the building blocks to model the different types of federation policies in a simple and intuitive fashion.

Policies are basically set of rules which can be interpreted automatically in order to control the behavior of a given system. Policies can be specified in several ways and various approaches have been proposed for different domains [2], [5], [12], [4]. The general requirements of any policy language have been highlighted in [16] and are as follows: *expressiveness, simplicity, enforceability, scalability* and *analyzability* to support reasoning for that language.

Rei [12], [11] is a policy framework that integrates support for policy specification, analysis and reasoning in pervasive applications. Its deontic-logic-based policy language allows users to express and represent the concepts of *rights, prohibitions, obligations* and *dispensations*. It also allows users to extend the basic ontology with additional domain-dependent

ontologies to express concepts and resources that are peculiar to certain domains. Rei's policy rules are represented as Prolog predicates. Predicate *has(subject, Policy Object)*, for example, defines the association between the *subject* of the policy and a *Policy Object* with the help of the *has* construct. The Rei framework provides a policy engine to reason about the policy specifications. The engine also accepts domain-dependent information which are expressed using triples of the form (subject,predicate,object). Since Rei is domain independent, it provides a very generic policy language. Thus, specifying federation policies in Rei can be very complex and require the use of a high number of language constructs. It is also not obvious if all our assertions can be expressed in Rei and how the history of transactions in a federation can be traced. In contrast, our language is very specific to the federation context and it provides a direct description of the history of activities in the federation. Also, through our logging system, we can efficiently control the user identity information flow and infer data without the need of an ad hoc inference engine. We are not aware of any specific feature in Rei for managing identities of users. As part of future work, we plan to explore how Rei can be extended to incorporate the assertions presented in our language. We will also explore the possibility of exploiting the domain-independent ontology provided by Rei for our assertion based language.

Ponder is a declarative object-oriented language that supports specification of several types of management policies for distributed object systems and provides structuring techniques for policies to cater for the complexity of policy administration in large enterprise information systems [4]. The main types of policies in Ponder are *obligations* and *authorizations*. Basic and composite policies in Ponder are distinguished in the following manner. On one hand, a basic policy is a rule specified by a declaration between a set of subjects (like users) and a set of targets (like resources or SP's). On the other hand, composite policies allow the basic policies relating to organizational units to be grouped to establish grouping constructs like roles and relationship policies. This provides a flexible mechanism to define subjects and targets in terms of domain scope expressions such that the combination of domains for a composite set of objects, at the same time be able to identify a single names object within single domains. Ponder provides a complete deployment model. At the end of the policy specification the policy is compiled by the Ponder compiler into a Java class and then represented at runtime by a Java object. The concept of management domains in Ponder befits the main concept of single federation and multiple federations. However to describe the federation properties and the varied policy types it would require including the comprehensive set of assertions presented in this paper.

## VII. CONCLUSION

In this paper we have developed a comprehensive set of assertions which is specifically relevant in the context of federations. Our assertions provide an intuitive approach to model federation activities and make access control decisions

based on a large variety of information, including past access history. We analyze the history of the behavior of entities and events with the help of an assertion audit log and query processing, and also provide a simple approach to specify policies. An important aspect we will work on concerns the definition of the protocols for resources exchange among the interacting parties. We will also further elaborate on the architectural framework supporting our approach to distributed logs.

## REFERENCES

- [1] A. G. Amir Herzberg. Trustbar: Protecting web users from spoofing and phishing attacks.
- [2] J. F. Barnes and R. Pandey. CacheL: Language support for customizable caching policies. In *Proceedings of the 4th International Web Caching Workshop*, 1999.
- [3] J. Chomicki and D. Toman. Temporal logic in information systems. In *Logics for Databases and Information Systems*, pages 31–70, 1998.
- [4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, 1995, 2001.
- [5] J. A. Hoagland. Specifying and Implementing Security Policies Using LaSCO, the Language for Security Constraints on Objects. *ArXiv Computer Science e-prints*, page 3066, Mar. 2000.
- [6] <http://shibboleth.internet2.edu>. Shibboleth, internet2.
- [7] <http://www.incommonfederation.org>. Incommon federation.
- [8] <http://www.oasis-open.org/committees/security>. Security assertion markup language specification set.
- [9] <http://www.projectliberty.org>. Liberty alliance project.
- [10] <http://www.w3.org/TR/P3P> preferences. P3P Preference Exchange Language .
- [11] L. Kagal. Rei : A Policy Language for the Me-Centric Project. Technical report, HP Labs, September 2002. <http://www.hpl.hp.com/techreports/2002/HPL-2002-270.html>.
- [12] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment, 2003.
- [13] G. Ozsoyoglu and R. T. Snodgrass. Temporal and real-time databases: A survey. *Knowledge and Data Engineering*, 7(4):513–532, 1995.
- [14] J. Park and R. Sandhu. Towards usage control models: beyond traditional access control. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 57–64, New York, NY, USA, 2002. ACM Press.
- [15] A. B. Spantzel, A. C. Squicciarini, and E. Bertino. Integrating federated digital identity management and trust negotiation. In *In review IEEE Security and Privacy Magazine*, 2005. CERIAS TR 2005-46.
- [16] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of kaos, rei and ponder. In *Proceedings of the International Semantic Web Conference, Sanibel Island*, October 2003. Invited survey.