Whom to trust? – Generating WS-Security Policies based on Assurance Information

Ivonne Thomas
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
ivonne.thomas@hpi.uni-potsdam.de

Robert Warschofsky
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
robert.warschofsky@hpi.uni-potsdam.de

Christoph Meinel
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
meinel@hpi.uni-potsdam.de

***Abstract*—As input for authorization decisions as well as to offer personalized services, service providers often require information about their users' identity attributes. In open identity management systems, these identity attributes are not necessarily managed by the service providers themselves, but by independent identity providers. Users might be required to aggregate identity attributes from multiple identity providers in order to meet a service's needs. On the other hand service providers might also have certain requirements concerning their confidence in these attributes and face the problem of choosing one among multiple identity providers that can possibly assert the same attributes, but with different trust qualities.**

In this paper, we present an architecture to generate service policies using assurance information about available identity providers. Our logic-based attribute assurance library, called IdentityTrust, allows the configuration of a knowledge base reflecting a service provider's knowledge about remote identity providers. Service providers can state their trust requirements concerning technical and organizational details of identity providers and their ability to assert identity attributes. A reasoning engine finds suitable (combinations of) identity providers, which serve as input for our policy framework that generates corresponding policies using the WS-Security policy format.

***Keywords*-Identity and Attribute Assurance, Identity Federation, Trust**

I. Introduction

In open environments such as Service-oriented architectures or the Web, participants as users, service providers and service consumer often do not know each other, but nevertheless require information from each other to perform meaningful transactions. Open identity management models have been designed to specifically address these needs. Identity providers are at the heart of these models and are set up by organizations and companies world-wide to share identity information in a controlled manner across organizational borders.

Naturally, these identity providers hold different sets of users' identity attributes with possibly different trust qualities. Companies, for example, might provide security token services for their employees to collaborate more effectively with partners. Such security token services that are offered by organizations can keep track of users in a much better way than identity providers for the web, which do not restrict their service to a specific group of people. In fact, with the latter ones often the registration of users takes place online without any verification. Nevertheless, both kinds of identity providers are useful and it depends on the intended transaction and its trust requirements which one is the best to be used.

Policies that are deployed with a service announce the trust requirements a service has and are essential to communicate with clients. With regard to identities, policies usually state required attributes (claims) as well as trusted identity providers the service accepts security tokens from.

In order to choose the right identity provider(s) for a given transaction, identity assurance is a major consideration. Identity assurance denotes the degree of confidence a service provider can have in the belief that a user's identity in the digital world actually matches with his real-life identity. Unfortunately, this confidence depends on many factors. Basically the whole process, technologies, protections, infrastructure and other safeguards in place at a remote identity provider, on which the provider's assertions are based, need to be taken into consideration.

In order to ease this process, we developed an attribute assurance library, called IdentityTrust, that allows to reflect these differences of identity providers. In particular, our framework can be used to describe for each identity provider not only whether it is trusted or not, but also which of the attributes that an identity provider holds are verified.

In this paper, we present an architecture that uses this library to generate policies. The benefit is, that a policy administrator can state requirements for attributes and identity providers with demands for a certain assurance level, which are automatically resolved to suitable (combinations of) identity providers and corresponding policy alternatives.

The rest of the paper is structured as follows. The following section describes a small motivating scenario that highlights our vision. Afterwards, Section III considers related work in the field of identity assurance. Section IV describes theoretical concepts before Section V describes

978-0-7695-4536-3/11 $26.00 © 2011 IEEE
DOI 10.1109/ECOWS.2011.29

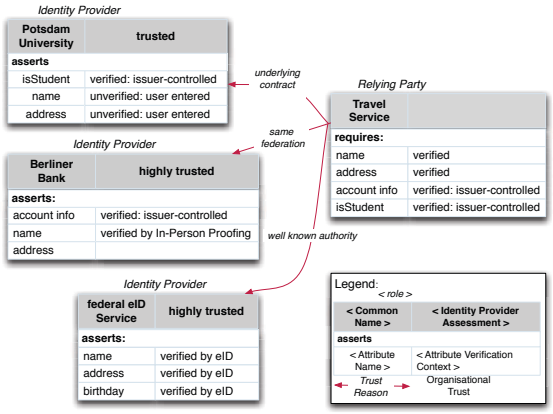

Figure 1. Vision of a scenario with multiple identity providers issuing identity attributes with different qualities of trust.

our architecture to generate policies from trust requirements that involve identity assurance aspects. Finally, Section VI concludes the paper.

II. Motivating Scenario

Our vision is an open world, in which identity providers are commonplace and users can choose freely between them to manage their identity information and to send it to service providers that require this information. As service providers can have widely varying requirements for identity attributes and their quality, users face the task to choose among many identity providers the one(s) that match with the service providers' needs.

In our vision, which is illustrated in Figure 1, each identity provider holds a set of attributes with possibly different qualities of trust. A bank could for example reliably assert a user's payment details while a university could assert whether a user is a student (cf. Fig.1: verified attributes). Also, with regard to the recent launch of electronic ID cards in various countries, an identity provider, which has been approved to access the users' electronic ID cards, can assert reliably attributes such as name, address or date of birth (cf. Fig.1 "verified by eID").

Service providers on the other side take the role of the relying party. Depending on the intended transaction, a service provider might require different degrees of assurance of a user's identity and its attributes. In our use case, which is used to illustrate our concepts throughout this paper, the relying party is a travel agency that offers various services on the internet. One of those services is a flight booking service, allowing a client application to make a flight booking. In order to do so, the booking service requires identity attributes such as name and address to issue the electronic flight ticket. Furthermore, if the user can prove that s/he is a student, 10 % is taken off the ticket price. Also, valid payment details are required to guarantee the booking. Regarding the confidence in the required attributes, we assume the following: For name and address any identity provider who verifies this data against an authoritative document is accepted by the service provider; whether the user is a student, can only be asserted by its university and finally for the payment details (account info), we assume, that the service provider accepts only identity providers that are part of the federation it is affiliated with, and that have verified their users' payment details.

Given such a list of trust requirements, this paper shows how we can derive policy statements that are expressible using existing policy web service standards, in particular WS-Policy[1] and WS-SecurityPolicy[2].

III. Related Work

The problem of identity assurance has been addressed from researchers and governments and international organizations alike.

A. Identity Assurance Frameworks

Several initiatives around the world have defined identity assurance frameworks with the aim to unify and standardize the perception of assurance into a digital identity. Among them are government initiatives (cf. for example UK Office of the e-Envoy [3], NISTs Special Publication 800-63 "Electronic Authentication Guideline" [4], etc.), as well as – in particular in recent years – also efforts from industry and the educational sector. One popular example is the InCommon Identity Assurance Assessment Framework [5] that is developed by the InCommon initiative [6], a federation of companies, universities and government organizations. The framework defines two different trust levels: *Bronze* and *Silver*. The Bronze level provides reasonable assurance that the same person is authenticating on subsequent visits. The Silver level builds upon the Bronze level requirements and demands for example stronger credential technologies, individual identity proofing as well as certain requirements to secure the business and IT security management processes that manage the digital identities. The Silver level is intended for medium risk transactions that require individual user accountability.

Compared to our approach, that provides assurance on the granularity of identity attributes, existing assurance frameworks mostly refer to the identity as a whole, and do not refer to trust requirements of specific attributes. It is for example not possible to distinguish between self-asserted attributes an identity provider might manage besides attributes that were verified. Also, using existing assurance frameworks, it is hard to reflect possible changes of a user's identity trust level over time. As identity proofing processes are cost-intensive and time-consuming due to the effort required to verify a users identity attributes, a verification

of an attribute might not be desired as long as a user is not involved in critical transactions.

B. Academic Approaches

Mohan et al. [8] provide a framework, called AttributeTrust, for evaluating trust in aggregated attributes. Attributes are provided by trusted attribute providers. Trust in these attribute providers in turn is calculated by using a reputation system model. In their approach, service providers express their confidence in other entities to supply trusted attributes. After each successfully completed transaction, service providers are asked to provide feedback for attribute providers. Over time, chains of confidence are formed between service providers and attribute providers. Compared to our approach, AttributeTrust builds up a global graph by aggregating feedback of multiple parties to express the confidence in other entities to assert trusted attributes. For this reason, as opposed to our work, individual assessments, based for example on the fact that a relying party has a contractual relationship to an attribute provider, are hard to express. Also, Mohan et al. do not differentiate between trust in attributes and trust in the attribute provider itself.

Work regarding trust levels for attributes has also been conducted by Chadwick et al. in [9]. Chadwick et al. build on NIST's [4] concept of assurance levels. Similar to our work, they propose to have separate metrics for identity proofing processes (expressed in the Registration LOA) and the authentication of a subject (expressed in the Authentication LOA). Authentication LOA and Registration LOA are combined to a Session LOA and sent in each assertion from an identity provider to a service provider. Compared to this, in our work, we assign trust levels not only to the registration and authentication process, but to individual attributes. For this purpose, we define verification context classes (cf. [7]) in a hierarchical manner to express details on the verification of attributes at an identity provider. Our motivation is to provide more choices for a relying party to express its demands and for identity providers to express their offers.

In her PhD thesis, Bharghav-Spantzel [11] establishes the notion of two types of assurance, namely validity assurance and ownership assurance. Validity assurance refers to the correctness of information while ownership assurance refers to the confidence that a claim actually belongs to a Subject. Her work is part of the VeryIDX [12] identity management system that fosters anonymisation techniques such as the *Zero-knowledge proof of knowledge (ZKPK) protocol* to preserve the users' privacy. Assurance levels are used to describe the ownership and consistency of identity records. Closely related with this work, is an approach for a quantitative definition of assurance levels for digital identities by Yong and Bertino [13]. In their paper, the authors take additional identity information as affiliation and social background into account in order to assess the assurance of identity records.

IV. Generating policies based on assurance information

Looking at the process of policy definition from the identity management point of view, the resulting WS-Security policy document states required claim types, token types as well as accepted issuers (if there are specific ones). In our research, we aim at supporting this policy definition process by resolving more general trust requirements concerning identity providers and attributes to concrete policy alternatives. Examples include contractual relations or federation affiliations, the verification of attributes as well as details about the registration of users. The foundation is provided by our attribute assurance library that allows a) to define a common assurance knowledge base and b) to reason over this knowledge base based on a given query. The output is a list of identity providers or combinations of identity providers matching the query. This list serves as input to create policy alternatives based on our policy model described in Section IV-B which can be translated directly to the WS-Security format.

A. Underlying Assurance Model

Our attribute assurance model consists of certain concepts such as identity providers $\mathcal{I}$, attribute types $\mathcal{T}$, verification context classes $\mathcal{V}$, etc. as well as relations among them and properties that apply in particular to the concept of identity providers as for example *isFederationPartner*$(i) = true$ for $i \in \mathcal{I}$.

The whole attribute assurance library is based on predicate logic, in particular on Horn clauses, as we found it to be a well-suited way to express facts, rules, and queries about identity providers, their trust relations, and security mechanisms in place. Facts are definite horn clauses consisting of exactly one positive literal, therefore exactly one predicate p. Rules are horn clauses consisting of a disjunction of one positive and at least one negative literal. They are usually written as implications:

$$p_1 \leftarrow p_2 \wedge \cdots \wedge p_n.$$

p_1 is called the *head* of the rule and $p_2 \wedge \cdots \wedge p_n$ is called the *body* of the rule. Facts and rules form together the *knowledge base*. Queries can be formulated in order to reason over the knowledge base. A query is a conjunction of positive literals. It has the form

$$p_1 \wedge p_2 \wedge \cdots \wedge p_n.$$

Knowledge Representation: All assurance knowledge is presented as facts and rules and stored in a knowledge base. Both, a centralized knowledge base as well as multiple decentralized knowledge bases are possible. A centralized knowledge base has the advantage that it contains the

aggregated knowledge of all participants. However, often it is not clear who can take the role of the trusted holder of the database. In the decentralized setting, each knowledge base reflects the subjective assessment of the holder, which might not be as complete as a centralized solution, but maybe more suitable in a decentralized environment. Putting it in a formal way:

- A knowledge base $\mathcal{K}_\mathcal{E}$ is a set of facts $\mathcal{F}$ that are trusted by the one(s) relying on this knowledge base: $\mathcal{K}_\mathcal{E} = \{j | j \in \mathcal{F} \wedge j$ is trusted by all $e \in \mathcal{E}\} \subseteq \mathcal{F}$. A knowledge base can be shared by multiple participants (entities) $\mathcal{E}$, that is $\mathcal{K}_{e_1,..,e_n}$ and $e_1, .., e_n \in \mathcal{E}$. It can also belong to a single entity, such as a single relying party: $\mathcal{K}_r$ for $r \in \mathcal{R} \subseteq \mathcal{E}$.

Facts and rules are predicates. The IdentityTrust library supports a number of pre-defined predicates as well as the definition of custom predicates:

Let $\mathcal{I}$ be a set of identity providers, $\mathcal{T}$ be a set of attribute types,

- $Assert : \mathcal{I} \mapsto 2^\mathcal{T}$ denotes that an identity provider is able to assert certain user attributes of a certain attribute type $t \in \mathcal{T}$, e.g. *Assert* ("Potsdam University") = {"isStudent", "name", "address"}. The corresponding predicate *Assert*(i_1, t_1) evaluates to *true* if $i_1 \in \mathcal{I}$ asserts attributes of type $t_1 \in \mathcal{T}$.

In order to characterize the verification of an attribute, verification context classes can be defined. The idea of verification context classes is to describe general verification schemes that are applicable to a set of attributes as described in Thomas et al. [7]. Examples include the verification based on a certificate (e.g. for email addresses) or on an authoritative document (e.g. name and address) both electronically or in-person. Various verification classes can also be subsumed under another verification class. Let $\mathcal{V}$ be a set of verification classes:

- *subClassOf* : $\mathcal{V} \mapsto 2^\mathcal{V}$ denotes that $v_1 \in \mathcal{V}$ is a sub class of another verification context class or of multiple other verification classes. For example *subClassOf*(*"checked by InPerson Proofing"*) = {*"verified"*}
- *Applies* $\subseteq (\mathcal{V} \times \mathcal{T})$ denotes that a verification class is applicable to an attribute type
- *AttrTrust* : $(\mathcal{I} \times \mathcal{T}) \mapsto \mathcal{V}$ where $(\mathcal{I} \times \mathcal{T}) \in$ *Assert* maps attributes hold by an identity provider to a verification class.

Furthermore, each identity provider $\mathcal{I}$ is characterized by a set of properties $\mathcal{P}$.

- A property $\mathcal{P}$ is a triple $\langle n, pr, v \rangle$, where $n \in N$ is a property name, $pr \in Pr_n$ is a binary predicate and $v \in V_n$ a value from the set of possible values for n.
- The following predicate symbols are considered: $Pr = \{=, \neq, <, >, \geq, \leq\}$. $Pr_n \subseteq Pr$ is the set of predicates that is applicable to a given $n \in N$.
- $hasProperty \subseteq (\mathcal{P} \times \mathcal{I})$ maps a property to an identity provider.

For example, in order to express the requirement, that the identity provider should be run by a university and that this university should be certified by an InCommon [14] trust level, we define the following identity provider properties:

$$N = \{n_1 = \textit{isUniversityIP}, n_2 = \textit{hasInCommonTrustlevel}\}$$

$$V_{\textit{isUniversityIP}} = \mathbb{B}$$
$$V_{\textit{hasInCommonTrustlevel}} = \{\textit{"none"}, \textit{"Bronze"}, \textit{"Silver"}\}$$

$$Pr_{\textit{isUniversityIP}} = \{=, \neq\}$$
$$Pr_{\textit{hasInCommonTrustlevel}} = \{=, \neq, <, >, \geq, \leq\}$$

Rules: Rules are basically derived facts, and can contain all pre-defined as well as custom predicates. A relying party could for example use a rule to define that an identity provider is trusted, if it has an InCommon [14] trust level of at least *Bronze* and its place of venue in Germany. Let $N = \{n_1 = \textit{placeOfVenue}, n_2 = \textit{hasInCommonTrustlevel}\}$ be defined properties to characterize available identity providers $i \in \mathcal{I}$ with reasonable values V. This would result in the following rule:

- *isTrusted* $(i) \leftarrow$ *hasInCommonTrustlevel*$(i) \leq$ *Bronze*$\wedge$ *placeOfVenue*$(i) =$ *Germany*

Queries: Queries are used to reason over the knowledge base. As said above, queries have the form $p_1 \wedge p_2 \wedge \cdots \wedge p_n$. To find for example, all identity providers, that can assert the attributes *name* and *address* in a verified manner, we formulate the query: *AttrTrust*$(i, \textit{name}) =$ *"verified"* $\wedge$ *AttrTrust*$(i, \textit{address}) =$ *"verified"*

B. Our Policy Model

To simplify the creation of service security policies a platform-independent policy model is described in [15]. This model serves as an abstraction layer for security policy languages, simplifies the handling of security policies, and enables the generation of security configurations in different policy languages.

A policy in this platform-independent policy model consists of several *Policy Alternatives* that contain a list of *Security Constraints*. In general, a *Security Constraint* describes a requirement to fulfill a *Security Goal* – such as confidentiality, integrity, or authentication – and contains information describing what has to be secured and which security mechanisms must be used.

A *Credential* is, as shown in Figure 2, used to describe parts of a digital identity. Therefore, a *Credential* contains a set of *Claims*. A *Claim* is "a statement made about a client, service or other resource (e.g. name, identity, key, group, privilege, capability, etc.)"[16]. It is required that for each *Claim* the corresponding information is added

to the *Credential* in the secured message. Additionally, a *Credential* can contain *Authentication Information* that can be used to verify the authenticity of the *Credential*.

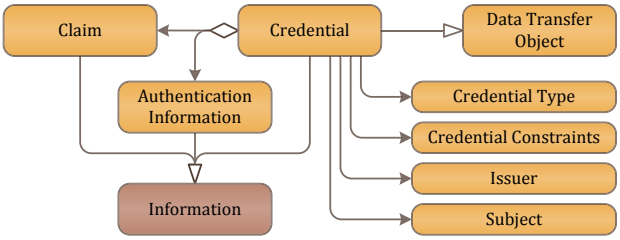

Figure 2. Digital Identity Meta Model

Each *Credential* also has a *Credential Type* that indicates its type (e.g *Username Token*, *SAML Assertions*, or *X.509 Certificate*). In addition, *Credentials* can have optional *Credential Constraints* that are used to further define the credential depending on its type. For example, it can be defined that a *Credential* of the type *Username Token* has to contain the users' password in a hashed form instead of plain text. Finally, each *Credential* has an issuer, identified by a unique URI. Depending on that issuer a receiver of the credential can decide whether the credential is trustworthy. The *Subject* element represents the subject of the *Credential*. For instance, the subject of a *Credential* of the type *Username Token* would be the user identified by the username. An example is shown in Figure 3. The *User Authentication Constraint* contains a *Credential* which requires that the sender of a message adds information about his identity to the message. Therefore, this constraint references a list of required claims and an issuer of the identity information.

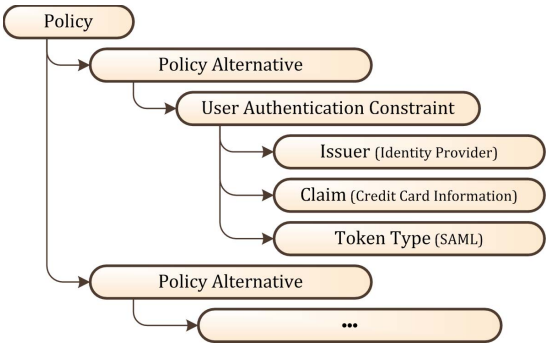

Figure 3. Policy Model Exmaple

V. Architecture

In this section, we present our architecture to generate WS-Security policies from trust requirements that contain assurance needs. We illustrate by means of our motivating scenario from Section II how our attribute assurance library

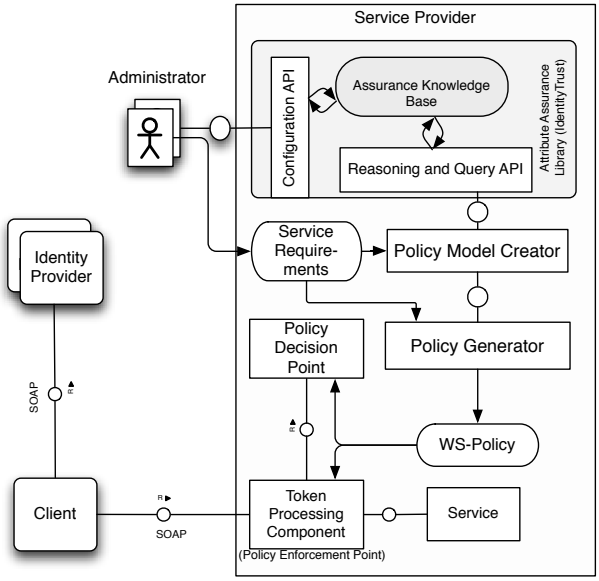

Figure 4. Architecture Overview.

can be used to define demands of services for identity attributes and match them with the offers of identity providers.

The result of our matchmaking process is a list of suitable (combinations of) identity providers that is translated by the *Policy Generator* (cf. Figure 4) to a policy document according to WS-Policy [1] that can be deployed with the service.

Figure 4 shows the architecture behind this process. There are three different roles involved in the exchange of identity information, namely the identity provider, the client and the relying party that provides the service. Generating policies and deploying them takes place at the relying party site. As can be seen from the figure, the relying party uses a domain-central knowledge base that is configured by an administrator or owner of the domain via the *Configuration API* provided by the IdentityTrust library. This knowledge base holds all the relying party's knowledge about other domains and their identity services.

The *Policy Model Creator* creates a platform-independent policy model as described in Section IV-B. The policy model retrieves as input the defined service requirements that are then transformed to policy alternatives by a matchmaking process that matches requirements for attribute claims with suitable identity providers by querying the knowledge base via the *Reasoning and Query API*.

The *Policy Generator* gets the platform-independent policy model and translates it to a specific policy language. The resulting document can be deployed directly at the service.

During service request, the received token is intercepted

and processed by the *Token Processing Component* and matched with the deployed policy document. Upon successful validation, access is granted to the *Service*.

A. Defining the knowledge base

The first step in setting up a knowledge base for a domain is to define the vocabulary that can be used in facts, rules and queries. As described in Section IV-A, a number of predefined predicates form a standard vocabulary to describe known identity providers, the claims they can issue as security tokens as well as the verification model that has been used. Beyond that, additional properties of identity providers can be defined, in particular concerning organizational and business factors, their technical infrastructure or the type of tokens they can assert. Typically, the owner of the domain decides which assessment criteria he considers. This can happen with the help of existing criteria catalogues, that are publicly available. InCommon for example provides a very comprehensive catalogue of assessment criteria that covers business, policy and operational factors, identity registration, credential technologies, identity information management as well as security management criteria. One option is to rely on such standardized assessments and define predicates as *hasInCommonTrustlevel* = {*"none"* | *"Bronze"* | *"Silver"* } or *isISOCertified* = {*true* | *false* } and add corresponding facts as *hasInCommonTrustlevel*(*"Potsdam University IP"*, *Bronze*) = *true* to the local knowledge base.

Besides such aggregated assessments, another option is to define the vocabulary in a more fine grained and technical way. Again, existing frameworks can serve as a base to find the right terminology. Taking for example the used electronic credential technology, InCommon defines the following assessment criteria: 1.) whether the credential uniquely identifies a Subject or 2.) the resistance of shared secrets to guessing. If a policy administrator wants to include these criteria in its trust decisions, s/he can easily define corresponding predicates as *UniqueCredentialIdentifier*={true | false} and *ResistanceToGuessingSharedSecret*={true | false}, and use them to create facts, rules and queries about other identity providers.

And finally, it might also make sense to add domain-specific properties to the vocabulary. For example, predicates as *isInTheSameFederation* or *isBusinessPartner* allow to tailor the knowledge base according to the current trust relationships and contracts of the owning company or organization.

B. Defining Service Requirements

Services usually have varying trust requirements depending on the risk that is associated with a transaction. A risk analysis reveals the required level of trust that serves as basis for the owner or administrator of the service to define an appropriate policy.

Table I
EXAMPLE REQUIREMENTS OF THE TRAVEL SERVICE

Attribute (Claim)	Assurance Requirements
name	*AttrTrust* (i, *name*) = *"verified"*
address	*AttrTrust* (i, *address*) = *"verified"*
account info	*AttrTrust* (i, *account info*) = *"issuer-controlled"* $\wedge$ *FederationPartner*(i) = true
isStudent	*AttrTrust* (i, *isStudent*) = *"issuer-controlled"* $\wedge$ *isUniversityIP*(i) = true

As described in [15], statements and requirements appearing in security polices can be categorized into basic correlation statements, security constraints expressions and digital identity expressions. While security constraint expressions define requirements about the protection of messages, digital identity expressions define requirements about needed verifications of the requester's identity. Verifications of the requester's identity can be expressed using the defined vocabulary of the knowledge base. If we take for example our motivating scenario in Section II, Table I shows the requirements for the *Travel Service*.

C. Policy Generation

Once defined, these requirements are used by the *Policy Model Creator* to create a platform-independent policy model that can be translated directly to the WS-Security policy format. As we are dealing with identity assurance, we are mainly focussing on requirements concerning the identity of users. Nevertheless, the resulting independent policy model can also contain additional requirements.

Querying and reasoning over the knowledge base: Our goal in this step is to formulate policy alternatives with *User Authentication Constraints* as shown in Figure 3 of Section IV-B. In terms of our assurance model, this means, we need to find for each required attribute suitable identity providers that match with our trust requirements. In order to do so, the *Policy Model Creator* creates queries over the knowledge base.

Our architecture is mainly implemented using Java-based frameworks; and also, the interface of the *Reasoning and Query API* of our IdentityTrust library is written in Java. A `Query` - object is created by the *Policy Model Creator* for each query to the knowledge base. Internally, our IdentityTrust library uses a Prolog engine to reason over the knowledge base and translates the query to a corresponding Prolog query, that is send to the Prolog engine. To give an example, the Prolog query to retrieve identity providers for the account information attribute is the following:

```
attrTrust( I, 'issuer-controlled',
     'account info' ),
federationPartner(I,true)
```

The corresponding result set in Prolog is:

1: $X \leftarrow$ Set of PolicyAlternatives
2: $I \leftarrow$ Hashmap containing for each claim a set of IPs
3:
4: invert I to a Hashmap C containing for each IP a set of corresponding claims
5:
6: **repeat**
7: $C_{max} \leftarrow$ set $C_x \in C$ where max($\#C_x$)
8: **for all** claim c in set C_{max} **do**
9: create policy alternative and add to X
10: remove c from all sets in C
11: **end for**
12: **until** all sets in C = {}

Figure 5. Algorithm to generate policy alternatives.

```
I='Berliner Bank IP'
```

Hence, the result of all queries is a set of sets of identity providers over all attributes: $I_{Result} = \{I_{account\ info}, I_{name}, I_{isStudent}, ...\}$. The *Policy Model Creator* uses a simple basic algorithm to extract from all result sets a combination of identity provider such that all required claims are resolved. This algorithm is shown in Figure 5.

WS-SecurityPolicy Generation based on our policy model: In order to create a WS-SecurityPolicy instance, the resulting combination of identity providers and the claims they will be requested for, have to be transformed into alternatives of our policy model. Therefore, for each identity provider in the combination, a policy model *Credential* is created which contains the claims and the identity provider as *Issuer*. An example of such a *Credential* is shown in Figure 6. If there are multiple combinations identified by the *Policy Model Creator*, for each combination a policy model *Alternative* is created and the corresponding *Credentials* are added to this alternative. Finally, the policy model is transformed to a WS-SecurityPolicy instance using the algorithm described in [15].

After the application of this algorithm, for each *Credential* a token in the `<SupportingToken>` section of the WS-SecurityPolicy instance has been created, similar to the `<SamlToken>`-element in Figure 6. If there are any other security requirements for this service, like the need to encrypt the messages to the service, they could also be stated using the policy model and would be transformed to the resulting WS-SecurityPolicy.

D. Calling the service

When calling the service, a service client needs to be able to deal with multiple policy alternatives in WS-Policy as well as requests for multiple issued tokens. While the specification supports multiple tokens, we experienced that the same does not necessarily hold for the implementations of the standards. In our use case, we use the Metro Web

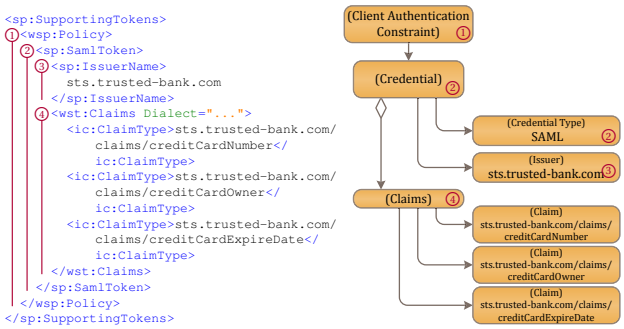

Figure 6. Transformation of a policy model *Credential* into a token in a WS-SecurityPolicy instance.

Service Stack [17]. While resolving a token request for a single token from one identity provider is not a problem, we needed to modify the Metro implementation slightly in order to allow request for two security tokens from different identity providers (two `<IssuedToken>`-elements) that are aggregated at the client side. In this case, the client reads the policy from our service, and sends a "request-for-security-token" first to the first identity provider and afterwards to the second one. Both received security token (SAML assertions) are added to the header of the SOAP message sent to the service.

The other case, that is to enable a client to deal with multiple policy alternatives (e.g. get a token asserting the address from either identity provider A or B), is not supported by Metro. Therefore, we implemented this logic directly in the client. Here, the client reads the policy from the service and tries to receive the required token(s) from the identity provider(s). If any of the request fails, the client starts over again using the next policy alternative. Upon successful retrieval of the required tokens, a request is send to the service including the security token(s) in its header.

VI. Conclusion

Past experiences with open identity management have shown that identity assurance is a critical factor for accepting information from another party outside one's own trust domain. Only if a relying party can assess the degree of confidence it can put in the assertions made by someone else, it will be able to accept this information and to use it in its own processes. For this reason, identity assurance plays an important role, in particular, when service policies are defined by policy administrators in the policy definition phase.

In this paper, we ease the process of defining a policy for a web service by providing a framework for identity assurance that allows the owner of a domain to a) describe its knowledge about partner and foreign identity providers in a knowledge base, b) to define service requirements that include assurance demands such as the verification of

attributes and c) to use the defined knowledge base to find suitable identity providers that fulfill these requirements. The approach is implemented in a Java-Prolog based library, called IdentityTrust, and can be used in various scenarios. In this paper, we present an architecture that uses the IdentityTrust library to generate WS-Security policies. The benefit of our architecture that has been implemented as a proof-of-concept is that the process of choosing identity providers according to certain assurance requirements is integrated in the policy definition process. Furthermore assurance parameters can be chosen in a more fine granular way.

REFERENCES

[1] Asir S. Vedamuthu et al.: Web Services Policy 1.5 - Framework, World Wide Web Consortium Recommendation, Sept. 2007.

[2] Anthony Nadalin et al.: WS-SecurityPolicy 1.2, OASIS Standard, July 2007.

[3] Office of the e-Envoy, UK: Registration and Authentication - e-Government Strategy Framework Policy and Guidelines, 2002.

[4] National Institute of Standards and Technology: Electronic Authentication Guideline, 2006.

[5] InCommon Federation: Identity Assurance Assessment Framework, 2010,
http: //www.incommonfederation.org/docs/assurance/InCIAAF 1.0Final.pdf

[6] InCommon Federation., http://www.incommonfederation.org/

[7] Ivonne Thomas, Christoph Meinel: *Identity Assurance In Open Networks* In Strategic and Practical Approaches for Information Security Governance: Technologies and Applied Solution, IGI Global, 2011.

[8] Mohan and Blough: AttributeTrust - A Framework for Evaluating Trust in Aggregated Attributes via a Reputation System. Sixth Annual Conference on Privacy, Security and Trust, pp. 201-212, 2008.

[9] Chadwick and Inman: Attribute aggregation in federated identity management. Computer (2009) vol. 42 (5) pp. 33-40.

[10] National Institute of Standards and Technology, http://www.nist.gov

[11] Bhargav-Spantzel, A.: Protocols and Systems for Privacy Preserving Protection of Digital Identity, PhD Thesis (2007). http://www.gradschool.purdue.edu/downloads/ETDForm9-E2.pdf, 1-222.

[12] Paci, F., Bertino, E., Kerr, S., Squicciarini, A., Woo, J. (2009). An Overview of VeryIDX - A Privacy-Preserving Digital Identity Management System for Mobile Devices. Journal of Software, 4(7), 1-11. doi:10.4304/jsw.4.7.696-706

[13] Yong, J., Bertino, E.: *Digital identity enrolment and assurance support for VeryIDX*. Computer Supported Cooperative Work in Design (CSCWD), 2010 14th International Conference on, 734-739. doi:10.1109/CSCWD.2010.5471880

[14] InCommon Federation: Identity Assurance Profiles Bronze and Silver, 2010.

[15] Robert Warschofsky, Micheal Menzel, Christoph Meinel: *Transformation and Aggregation of Web Service Security Requirements*, In ECOWS'10: Proceedings of the 8th IEEE European Conference on Web Services, pp 43-50, 2010

[16] *WS-Trust 1.3*, OASIS Std., 19th Mar. 2007. [Online] Available: http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.pdf

[17] *Project Metro* 2007. [Online] Available: http://metro.java.net/discover/