Semantic Policy Enforcement and Reconciliation for Information Exchange in XMPP

Brian Ulicny, Won Ng, Oleg Simakoff, Jakub Moskal
VIStology, Inc.
Framingham, MA USA
{bulicny, wng, osimakoff, jmoskal}@vistology.com

Mieczyslaw M. Kokar
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA USA
m.kokar@neu.edu

***Abstract*— Extensible Messaging and Presence Protocol (XMPP) is a popular open-standard protocol for instant messaging (IM) widely used in military and commercial applications. In military contexts, as in commercial settings, it is often necessary to regulate who may communicate with whom and how. The distributed nature of XMPP makes centralized information exchange policy enforcement impossible, however. We report on a technology we have developed, called PolVISor, in which we express information exchange policies in a natural language formalism (SBVR SE), automatically translate these policies into an executable rule language (BaseVISor rule language) and enforce and reconcile disparate policies among XMPP servers, each with its own policies, using semantic technologies.**

Keywords: XMPP; security policies; policy reconciliation; SBVR; ontologies; deontology; modality

I. INTRODUCTION

Policy authoring, representation and enforcement are essential components in security systems. As systems grow and collaboration becomes more ubiquitous (e.g. via grid computing, collaboration among coalitions), the set of security policies grows larger. This leads to potentially undetected policy conflicts and the need for automated or semi-automated policy reconciliation. Our work has resulted in PolVISor, which uses ontological reasoning to determine security policy compliance and provide policy reconciliation when possible. We demonstrated the necessity, feasibility and flexibility of PolVISor to constrain information sharing in an XMPP (Extensible Messaging and Presence Protocol) environment.

II. SECURITY, POLICIES AND RECONCILIATION

In this project we were concerned with the ability to use policies to ensure compliance during runtime as well as with the ability to do policy reconciliation. Policy compliance involves the run-time process of ensuring that all of the conditions defined by a policy hold true; a common example is the checking of credentials required before granting access to a document. In policy reconciliation, the goal is to take multiple polices and, e.g., generate a policy instance that simultaneously satisfies all of them; a typical example here is determining specific conditions under which a communication session can be established between nodes in a VPN where the ends of the connection are governed by different policies.

1.1 Semantic and Non-Semantic Representations of Policies

Policies can be implemented in a system via the hardware (e.g. this light will not turn on unless both of these switches are turned on); or in software. In software, a policy can be represented either syntactically or semantically. By a semantic representation, we mean a representation in which inferences can be made on the basis of a policy instance using a domain-generic inference engine. So, for example, a Windows Group Policy instance has a meaning that is clear to everyone who knows the semantics of the policy language. However, no generic reasoning engine can draw inferences from Windows Group Policy instances in their native format. The representation has no meaning to those engines.

A primary objective in our work is to develop the means by which operations governing policies can be handled automatically by a computer. For this reason it is important to be able to describe policies in a formal, declarative way that will permit them to be automatically processed by formal reasoning engines.

A formal reasoner or inference engine is a system capable of applying the formal axioms of a language to a body of data/facts/knowledge resulting in the derivation of additional inferable facts. A rule-based system, for example, may be used as a formal reasoner if it is provided with a set of axioms for the language in which the data/knowledge is represented. Such axiom sets are available for a number of ontology languages as discussed below.

An important principle employed by many systems including policy-based reasoners is the use of the closed world assumption (CWA), which permits systems to assume that everything that is known to be true of the "world" is available in the facts that have been provided about it; if a fact is not explicitly stated it is assumed to be false. The closed world defined by a set of facts can be thought of as a "context" in which reasoning is to occur. OWL-based systems, like PolVISor, do not adopt the CWA.

For reconciliation to be possible there should be an explicit separation of policies and mechanisms that use the

policies, and the policies should be first-class objects within the security system. In this way, policies will be objects that can be represented, stored and manipulated by the security system. Moreover, in this way policies will have their own interpretation, or semantics. This has a very important impact on the accreditation process in that mechanisms can be accredited and then policies can be added dynamically.

1.2 The Policy Reconciliation Problem

Two systems or elements of a system may impose policies on certain operations. In this paper we define policy reconciliation as the determination of a policy that implicitly or explicitly satisfies both policies and governs the behavior of the interaction of the system(s). Provisioning policies, authorization policies and information exchange policies are all types of policies that may require reconciliation.

In this project, we have bounded the problem of policy reconciliation in several ways. First, we assume that all partners in the policy negotiation process are equals. Therefore, we have chosen not to incorporate policy deference mechanisms saying that if System 1 and System 2 have different policies, then one of the system's policies overrides the other. While such meta-policies are widespread in practice, they do not pose an interesting conceptual problem.

Secondly, we have not dealt with preferences among policies. Thus, a system might allow distinct set of actions A or B (distinguished by their participants, say, or by other parameter settings), but it would prefer one set to the other. We have not addressed this issue because it essentially involves a different kind of modal reasoning: reasoning that ranks some situations as more desirable than others, although each is permissible. This is the logic of "should" and "should not", as opposed to the logic of "may (not)" and "must (not)" as described in the section on our deontic ontology of actions below. The considerations involved in modal reasoning about 'should' involves a higher-order reasoning than the logic of 'may' and 'must', and we have not addressed this in this project. In particular, we have not addressed what might be called "consequentialist" policies, where a policy is preferred based on its outcome. For example, one might say, choose policy A or policy B based on which one allows the most (or fewest) users (perhaps meeting some other criteria) to access some set of files. Such a system would require some kind of modeling and simulation step to determine how many users have access, and thus determine the policy choice.

Finally, we have not concerned ourselves with situations in which the policies to be reconciled cannot be completely disclosed between the interested parties. There are undoubtedly situations in which the policies that govern some action are themselves proprietary and sensitive in that they reveal, with contextual information, proprietary information. For example, suppose a University had a policy in which admitted students could sign up for a campus bulletin board system. If prospective students learned about this policy, they could potentially find out who had been admitted to the university before the official announcement had been made by trying to register on the bulletin board. In such a case, the university might want to avoid making such a policy known to other users or systems in order not to disclose unwanted information. We have not focused on such situations of policy reconciliation where trust is an issue since trust management is beyond the scope of our current investigations.

1.2.1 Information Exchange Policies

In this paper, we examine enforcing and reconciling information exchange policies. Information exchange policies are important in military and intelligence situations, where cross-organizational collaboration is required but strict policies restrict who can communicate with whom and what information they can exchange. For example, a military coalition might allow members of different national forces to collaborate on some tasks within certain channels and with certain information, but not others. The same is true of financial services and health care industries, which both regulate information exchange. For example, in financial services, so-called Chinese Wall policies regulate communication between analysts and traders. In health care, privacy and confidentiality policies regulate what information can be shared between health care providers and patients. Information sharing between social networking sites and other sites is another current example, particularly where single sign-on schemes like OpenID (http://openid.net) are involved.

In the military and intelligence community, information exchange policies are labeled "Cross Domain Solutions": "Cross Domain Solutions (CDS) are controlled interfaces that provide the capability to access or transfer information across different security domains." [1] The eXtensible Markup Language (XML) Data Flow Configuration File (DFCF) format specification[2] was developed to provide a common format for defining, validating, and approving XML data flows for use in XML cross domain solutions. DCDF is specified syntactically in XML in terms of information sharing system endpoints, where a complete policy specifies, for each endpoint pair, what information can be sent from an endpoint, and what information may be received by an endpoint. Such comprehensive policies are difficult to set up, are likely to become obsolete as the contents of the endpoint systems change, and are not flexible. Finally, they are not reconciled, across all endpoints because one system cannot impose any limitations on another system, only on itself. However, they can be implicitly reconciled at run time when two endpoints try to exchange information.

III. POLICY LANGUAGES

In our project, we use SBVR Structured English (SE) for authoring policies in an English-like formalism. SBVR SE policies are then automatically translated into BaseVISor Rule Language (BVR) for execution and policy reconciliation.

---

[1] Unified Cross Domain Management Office, What is a cross domain solution?, http://www.ucdmo.gov/faqs.html.

[2] XML Data Flow Configuration File Format Specification Version 1.2.11 19 December 2008 http://iase.disa.mil/cds/helpful_tools/dfcf-specification-1-2-11.pdf

1.2.2 SBVR Structured English

Semantic of Business Vocabulary and Business Rules (SBVR) [1] is an OMG standard introduced in 2008 that aims at a more natural format for expressing rules. Business rules are expressed in a subset of natural language that is readily understandable by business people, instead of at an implementation level, such as rules that are processable by a formal reasoning engine. The vocabulary represents the concepts used in the rules and can also express facts and relations between concepts (e.g. that Fido is a dog). The specification is based on first order modal logic and captures the semantics of implementation-independent business models. Figure 1 locates SBVR in the Business Model (also called the Computation-Independent Model) level in OMG's Model Driven Architecture (MDA) [2] and is meant to be translatable to a Platform-Independent Model (PIM) that describes the structure and behavior of the model, and subsequently to a Platform-Specific Model (PSM) that includes all the platform dependent information necessary for a developer to implement executable code, such as specific programming language packages. SBVR is mapped to the Meta-Object Facility (MOF) [3] metamodel – a useful feature for transformations of an SBVR model to other models.

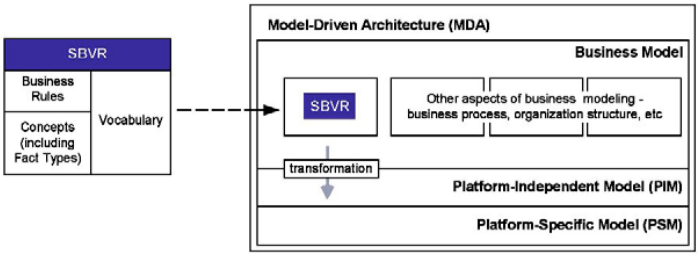

Figure 1: SBVR in OMG's MDA.

SBVR distinguishes between *alethic* and *deontic* constraints. Alethic rules are categorized as structural business rules, which are rules that must necessarily be true as part of the business organization. Deontic rules are operative business rules that should be obeyed but which can be violated in practice.

SBVR has two common notations: Structured English and RuleSpeak®. Structured English (SBVR SE) is a controlled English vocabulary and grammar that uses font styling and color to indicate SBVR concepts. term represents a noun concept such as rule and action. Name is an individual concept and usually is a proper noun, e.g. California. *verb* is part of a SBVR construct called a fact type and is usually a verb, preposition or combination of preposition and verb. Lastly, SBVR SE defines a set of keywords that are reserved words or phrases with special meaning. Examples of keywords are the articles a and the, modality phrases It is necessary that, and quantifications every and at most one. An example of a SBVR SE rule is:

It is obligatory that a driver *is qualified* if the driver *rents* a car that *is owned by* EU-Rent

SBVR RuleSpeak® [4] is a proprietary variant developed by Business Rule Solutions, LLC (BRS) [5]. RuleSpeak® provides templates for business rules based on the category or subcategory that applies to the rule. We did not use the RuleSpeak format and will not address it here.

Like the other languages discussed, SBVR is domain and application independent. The SBVR specification includes a proposal relating SBVR concepts to equivalent OWL expressions, so clearly some consideration was given to how SBVR should work with semantic languages. Its main strength over the other languages is its user friendliness. Because SBVR SE is an almost-natural language, it is suitable for expressing high-level rules. Among available editors are SBVR-VE (SBVR Visual Editor) [7], a graphical drag-and-drop editor where attempts to create links between boxes containing, say, a modality and a term, would often not work; Sepiax-Web [8], an Ajax-based web editor with WordNet and SBVR integration; SBeaVer [9], an Eclipse plugin that provides syntax highlighting for SBVR SE; and a proposed SBVR tool component [10], including an editor, as part of Eclipse's Modeling Development Tools (MDT) [11] that has been in development for the past few years. There are also enterprise editors that support RuleSpeak®.

SBVR is sufficiently expressive for representing high level rules but because SBVR is at the business model level, it suffers from the common problem that most business model level components do: translation to a PIM and especially to a PSM requires additional details about computations and platform-specific information, usually supplied by an IT person. The SBVR vocabulary can be expanded to include platform vocabulary, but SBVR is meant to be a high level language and is not executable, so SBVR is most useful when translated into a lower level executable language like BaseVISor Rule Language (BVR), as we have done.

1.2.3 BaseVISor Rule Language (BVR)

BaseVISor (http://www.vistology.com/basevisor), a versatile forward-chaining rule engine specialized for handling facts in the form of RDF triples (i.e., subject, predicate, and object), expresses rules in BaseVISor Rule language (BVR). The BaseVISor engine implements OWL 2 RL inference rules in BVR and supports XML Schema Data Types.

Generally speaking, rules are expressed in the form of if/then statements. The 'if' part of the statement is represented by the 'body' or 'antecedent' of the rule; the 'then' part is represented by the 'head' or 'consequence'. In BVR the contents of rule heads and bodies are made up of triple patterns (i.e., triples that may contain variables) and procedural attachments, i.e. functions such as *add*, *assert*, and *println* (print line). Users can add user-defined procedural attachments for use in rules. BaseVISor also supports queries, which are special cases of rules with empty heads, and are useful for retrieving information from the resulting fact base.

BVR is domain and application independent, compatible with the semantic languages OWL and RDF, designed for formal reasoning and executable in the BaseVISor environment. It is very expressive, especially since the language is extensible via user-defined procedural attachments. A BVR editor is available as an Eclipse plugin to aid in composing BVR rules.

Translation of SBVR SE into BVR makes use of metamodels for both languages. First, SBVR SE expressions of policies are saved as XMI, then a proprietary metamodel-to-metamodel mapping is used to translate the SBVR XMI into a corresponding BVR rule, preserving its semantics.

IV. Security Policy Ontologies

We developed two OWL ontologies to encapsulate our treatment of policies as classes and to represent concepts and their relations that we have determined to be essential for security scenarios, including information exchange. These "core" ontologies are the basis for any domain-specific application of PolVISor, i.e. domain-specific scenarios should extend these ontologies with their domain-specific knowledge and rules. The design of the ontologies, such as treating actions and operations as first-class entities, are grounded in our study and investigation of formal security models.

1.2.4 Representing Modal Notions in OWL

PolVISor, as we have said, involves two kinds of modality, *deontic* and *alethic*. Modal expressions qualify the truth of a statement. For example, to say that "John is possibly dyslexic" is not to assert that "John is dyslexic", but a more qualified statement that the statement might be true. Modality is expressed logically as operators over propositions. Op(p) means that some modal operator *Op* is being asserted of the proposition p: *It is Op that p.* The operator identifies the way in which the truth of the bare proposition p is being qualified.

Alethic modality is the logic of possibility (it is possible that p) and necessity (it is necessary that p). As specified by SBVR, alethic notions are encoded directly in the ontology. Necessity relations between classes are expressed in terms of subclass relations that apply to all instances. Thus, to say that "necessarily, all bachelors are unmarried" or "necessarily, all cats are mammals" is to say that the class Bachelor is a subclass of Unmarried Things and that Cat is a subclass of Mammal. Without such a subclass relation, it might be that all of the instances of Bachelor are instances of Unmarried, but that would be a contingent coincidence, not a necessary truth, with respect to that ontology. We encode that it is possible that (some) Fs are Gs (e.g. that some File Clerks are Dyslexic) in the ontology by failing to have class F (File Clerk) and G (Dyslexic) as disjoint classes. If F and G are marked as disjoint classes, then necessarily, no Fs are Gs, (and, necessarily, no Gs are Fs), according to that ontology.

"It is necessary that a user has a password" expresses a necessity relation between the class of Users and the class of things that have a password. This necessity relation would be expressed by saying that the class of Users is a subclass of the class of things that have Passwords. This encodes the necessity relation in the ontology directly. Ontologies, after all, express constraints on how the world can be. To say that users *may* have a password is expressible by saying that the class of Users and the class of things that have a password are not disjoint.

Deontic Logic [12] is the study of the logic of the concepts "may" (or deontic 'can') and "must" and their duals "may not" and "must not". These concepts are crucial in expressing policies: policies express what may or may not be done, under certain conditions, and what must and must not be done, again under certain conditions. *May* and *must* are modal notions. Sentences employing modal notions do not express the way the actual world is, but qualify the truth of the proposition they modify, in this case expressing conditions on how possible worlds should be if they are to comply with the policies our ontology encodes. That is, if I say that "John may go to the store" or "John must (not) go to the store", I do not say anything about how the actual world is with respect to John's going to the store. What I express has to do with the consistency of John's going to the store with the ways in which John is permitted to act or with the ways in which John must act.

In our inference engine, BaseVISor, propositions are expressed as triples (subject, predicate, object). BaseVISor does not allow for modal operators over triples. Therefore, rather than give modal operators their usual semantics as quantifiers over possible worlds or ways the world could be or ways a person could act, we treat Actions as a class that can be subdivided into Permissible (may), Omissible (may not), Optional (may and may not), Obligatory (must) and Prohibited (must not) subclasses.

The structure of the ontology is represented in Figure 2:

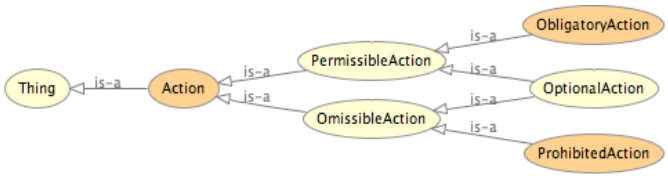

Figure 2. Classes and subclasses of Deontic Ontology

First, Actions are subclassified as Permissible or Omissible. An action is Permissible if it may be done. For example, getting married is permissible, so the class of actions that is getting married could be represented as a subset of the class of permissible actions.

An action is Omissible if it is permissible not to do it. For example, eating okra is omissible. One may abstain from eating okra. The class of actions that is okra-eating could thus be represented as a subset of the Omissible actions.

In fact, one both may and may not eat okra (and one may or may not get married), so instances of both of these types of actions would be instances of the intersection of the Omissible and Permissible classes: the Optional actions.

Obligatory actions (actions one must do) are a subset of the Permissible actions. If an action must be done, then it may be done. The Obligatory actions and the Omissible

actions are disjoint: if an action must be done, it is not the case that it may not be done.

Similarly, Prohibited actions (actions one must not do) are a subset of the Omissible actions (actions one may not do). The Prohibited actions and the Permissible actions are disjoint: if an action must not be done, then it is not the case that it may be done.

We have expressed these relations in an OWL ontology. The ontology may be downloaded at http://vistology.com/ont/2010/secpol/Deontic.owl.

By means of this ontology, one can state that all instances of actions of a certain type are, for example, prohibited (e.g. theft, murder) or permissible (e.g. expressing one's opinion, forming associations) across the board. Policy rules allow one to express conditions under which actions of a certain type are classified as permissible or prohibited or optional based on additional facts about them. For example, one could express the policy that it is permissible to marry only if one is at least a certain age, not already currently married, and so on.

1.2.5 Upper Policy Ontology

We developed a policy ontology to serve as the base of all application- or domain-specific ontologies, available at http://vistology.com/ont/2010/secpol/UpperSecPolOnt.owl. It was derived by starting with the Naval Research Laboratory's (NRL) Security Ontology [25]. The NRL ontology was primarily designed for annotating resources with security-related metadata in order to facilitate the discovery of resources that meet security requirements.

In our ontology, a Policy consists of one or more Rules, associated with a SecurityPurpose (e.g. Data Integrity, Confidentiality). Rules are expressed in SBVR SE and translated into BaseVISor rule language. Rules govern Operations (Actions), i.e. operations performed by an element in the system (e.g. reading a file). Each Operation has an agent who originates the operation and an object that is the target of the operation. In the example of Bob reading a file foo.txt, the operation is a Reading with agent Bob and object foo.txt.

These Operations are declared to be owl:sameAs the class of Actions in the Deontic ontology, and thus, subclassified as Permissible and Omissible, and so on. They can also be equated to some other ontological representation of operations. Here we assert our Operation class to be owl:sameAs the class of UCore-SL Acts, indicated by the namespace sl. UCore-SL is an OWL version of the UCore [13][14] messaging format adopted for information sharing among the defense and intelligence communities.

For Security Markings, we have employed Richard Lee's ISM Ontology v. 0.7 [15]. This ontology is described as "a rendering of the IC-ISM XML spec for security markings. It is based on the IC-ISM v5 XSD, updated thru 2010-09-25. Although this ontology provides a complete taxonomy of security markings in use by US and Coalition partners, it does not generally order security markings from high to low within a markup scheme. We have added axioms to encode these facts as needed.

V. INFORMATION SHARING IN XMPP

Extensible Messaging and Presence Protocol (XMPP) [16] is a popular open-standard protocol for instant messaging (IM) widely used in military applications. There are a number of extensions to the protocol that define protocols for other functionality, like Voice Over IP (VoIP). Each user signs into his XMPP account identified by a jid, commonly of the form *name@domain.server*, e.g. *juliet@montague.net*. Each jid has a contact list called a roster. Figure 3 illustrates the process when a user signs on. The server hosting the user automatically sends a presence to each of his contacts, except for those he has blocked, to indicate that he is now online. The contact's server forwards the presence to the receiver, unless she specified that she does not wish to receive presences from the sender. The contact's server also sends back a presence to the sender if she has not blocked presence-outs to the sender. Now the two clients can start chatting with each other. Users can also join chatrooms, participate in conversations as a group, and send messages to individuals in the room.

Privacy lists allow users to specify contacts with whom he wishes to restrict contact. However, there are currently no methods for server to specify policies to restrict users' chat, except by name. Using Openfire [17], an open source XMPP server available from Ignite Realtime [18], for our server, we developed an Openfire plugin that intercepts incoming and outgoing XMPP stanzas. The stanzas of interest in our scenarios are presences and messages, but all stanzas are intercepted so our implementation is extensible. Users connect to servers via Spark IM Client [19], an open source IM client application also provided by Ignite Realtime.

The Openfire plugin plays the role of the context handler here. It invokes an XSLT script to translate the XMPP stanzas to RDF and passes the RDF version of the stanza to PolVISor. PolVISor analyzes the stanza and returns to the plugin a decision to allow or deny the stanza. We chose to implement a deny-overrides approach, where if any applicable rule denies the stanza, the stanza is denied. If the stanza is allowed, the plugin forwards the original stanza to Openfire, which processes it as usual. If the stanza is denied, the plugin drops the stanza and the server does not see it. The behavior of the plugin can be changed to modify the stanza instead, for example if dropped stanzas should be logged by the server.

We developed an XMPP ontology with the core concepts such as jid and presence. The scenarios below build upon this base ontology. Because of our action-oriented approach in the upper ontology, actions like *Sends* are subclasses of Operation. BVR rules convert the stanza information to match the ontology, e.g. based on a presence stanza from *juliet@montague.net* to *romeo@capulet.com*, a *Sends* instance is generated that has agent the sender *juliet@montague.net* and has object the presence stanza, and the presence stanza has "to" *romeo@capulet.com* and "from" *juliet@montague.net*.

A. XMPP Presence Scenario

To demonstrate server policies that limit who can communicate with whom, based on facts about the persons involved, we implemented rules and ontologies for one server

that restricts chat based on gender and another server that restricts chat based on the first letter of the jid. Gender is used for simplicity, but any class of persons could be used here, for example, filtering users by any combination of role or nationality or location. Because chatting depends on the initial sending of presences to contacts, the rules analyze presence stanzas and apply to incoming and outgoing presences. The rules state:

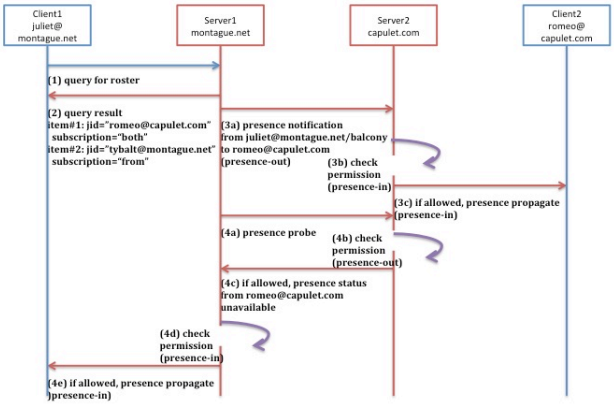

Figure 3: XMPP sequence diagram for sign in and roster retrieval.

Server 1 Policies:

Allow presences to/from Males on Mondays, Wednesdays and Fridays.

Allow presences to/from Females on Tuesdays, Thursdays and Saturdays.

Each user's gender is encoded using the FOAF (Friend of a Friend) vocabulary, and the information is available to Server 1. Because the FOAF gender is an untyped literal, a helper BVR rule determines whether a jid is an instance of the class Male or Female accordingly.

Server 2 Policies:

Allow presences to/from contacts whose jid start with A-L on Mondays, Tuesdays and Wednesdays.

Allow presences to/from contacts whose jid start with M-Z on Thursdays, Fridays and Saturdays.

Server 2's rules take advantage of BaseVISor's built-in regular expression procedural attachments.

All other stanzas that are not allowed explicitly are denied, e.g. no one hosted on either Server 1 or Server 2 can chat with others on Sundays. Here, policy reconciliation is implicit; a presence successfully sent from a user on Server 1 to a user on Server 2 means that both Server 1 and Server 2 allow the stanza. Therefore, amy@server1.com who is Female on Server 1 can chat with brenda@server2.com who is Female on Tuesdays because of Server 1's second rule and Server 2's first rule.

B. XMPP Security Labels Scenario

CWID 2010 featured a Cross Domain Collaboration implementation [20]. The collaboration scenarios included chatting and document sharing using security labels, access control and authentication. Clients and servers were modified to support security labels, among other functionalities. Boldon James's SAFE IM for XMPP [21] allows users to assign security labels to their one-to-one chats, group chats in rooms, and file transfers. It checks that receivers of labeled messages have sufficient clearance to read the message and that users who wish to join a chatroom with a security label have sufficient clearance to join. Isode's M-link server [22] is a XMPP server with support for controlling message flow based on the security label of the message and the security clearance of the sender and recipient.

We have implemented the same functionality of security labels by extending the ontologies and rules for the presence scenario outlined previously. Both sets of ontologies and rules for Server 1 and Server 2 have the same extensions and use the security levels from Intelligence Community Information Security Marking (IC-ISM) ontologies [23] for security labels and clearance levels. We added reflexivity and transitivity to the relevant properties so PolVISor can reason that someone with a clearance of TopSecret can send and receive messages classified as TopSecret or any lower level like Secret or Unclassified. This scenario considers one-to-one labeled chat messages but can be easily extended to group chat messages sent to a labeled chatroom so that no messages with a label at a higher level than the chatroom's maximum allowed label could be sent. Clients set the level by enclosing the label in brackets in the beginning of the message body, e.g. [RESTRICTED].

The rules state:

If a sender sends a labeled message to a recipient on a different server and the sender has equal or higher security clearance than the security level of the message, then the message is permitted to be sent.

If a recipient of a labeled message from another server has equal or higher security clearance than the security level of the message, the message is permitted.

If the sender and recipient of a labeled message are on the same server, and if the clearance of the sender and clearance of the recipient are equal or higher than the label's level, the message is allowed.

If a message does not explicitly have a security label, the message's security label is Unclassified.

All stanzas not explicitly allowed are denied.

C. XMPP Chatroom Reconciliation Scenario

To demonstrate explicit reconciliation, we implemented another scenario. If a client on Server 1 wants to join a chatroom hosted on Server 2 and both Server 1 and Server 2 have security policies restricting who can join what chatroom, then their policies must be successfully reconciled and the attempt to join must satisfy the reconciled policy in order for

the attempt to be allowed. By satisfying the reconciled policy, the request also satisfies each server's policy. A client joins a chatroom by sending a presence to the chatroom, so the rules analyze presence stanzas.

The rules state:

Server 1 Policy: If the client is Male, he can join any chatroom.

Server 2 Policy: Any client can join any chatroom.

Server 1's policy is more restrictive than Server 2's, and lacking any other rules that concern clients joining chatrooms, ensure that only Males are allowed in chatrooms that involve any Server 1 clients. Figure 4 depicts the process. The plugin and PolVISor are not explicitly shown, but rather are subsumed as part of the server. The reconciled policy in this case is logically equivalent to Server 1's policy since Server 2's policy subsumes Server 1's. Therefore, reconciling the policies is equivalent to adopting the more restrictive policy.

However, because the servers have their own extended ontologies with server-specific classes and properties, ontology mapping could be needed for the request to satisfy the reconciled policy. An ontology mapping scenario is discussed below.

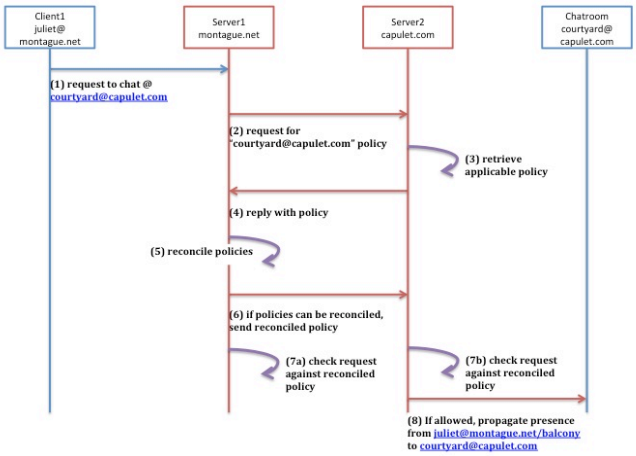

Figure 4: XMPP chatroom reconciliation sequence.

D. *XMPP Ontology Matching Scenario*

So far we assumed that both Server 1 and Sever 2 use the same ontology-based vocabulary to describe their clients. However, it is possible that the servers use facts expressed in different ontologies, in which case before polices can be reconciled, *ontology matching* must be first performed.

Ontology matching is the process of finding relationships between entities in two or more different ontologies. The output of matching, called an *alignment*, is a set of correspondences that express the relationship between different ontologies. Alignments include, but are not limited to, statements such as entity equivalence, sub-super relationship between entities, class intersection, or inverse relation. Alignments can be used to generate various tools used in further automated processing. For instance, a *translator* can translate data instances expressed in one ontology to another, or a mediator that can translate queries expressed in one ontology to another, and translate answers in the opposite direction.

Despite sophisticated methods from AI, ontology matching currently can rarely be fully automated beyond relatively simple correspondences, covering syntactic and terminological heterogeneity. When the same concepts in different ontologies are defined using different axioms, matching algorithms often have difficulties identifying any correspondences at all, or find ones that are irrelevant. When matching is incomplete or incorrect, manual editing is necessary. Matching systems typically allow the user to specify a threshold for confidence held in the correspondences, which allows for eliminating matches that are most likely invalid.

In order to demonstrate the use of automated ontology matching in the process of policy reconciliation, we matched FOAF and vCard ontologies with the threshold of 0.9 (1 being 100% confident) using an ontology matching API [24] and dynamically found the following relationships:

Equivalent classes:
- Foaf:Organization and vcard:Organization

Equivalent datatype properties:
- foaf:givenname and vcard:given-name
- foaf:givenName and vcard:given-name
- foaf:nick and vcard:nickname
- foaf:title and vcard:title
- foaf:family_name and vcard:family-name
- foaf:familyName and vcard:family-name

Equivalent object properties:
- foaf:logo and vcard:logo

Since vCard does not include gender information, we could not directly use this property to define policies. Instead, we used the nickname information, supported by both ontologies, in order to encode the gender of a user. We assumed that all nicknames follow a pattern "g_Name", where "g" can be either "F" or "M", indicating the user's gender. We designed a scenario similar to the XMPP Chatroom Reconciliation Scenario, with the following rules:

Server 1 Policy: If the client is Male (i.e. if his foaf:nick starts with M_), he can join any chatroom.

Server 2 Policy: Any client can join the chatroom.

Reconciled Policy: If the client is Male (i.e. if his foaf:nick or vcard: nickname starts with M_), he can join any chatroom.

While the policies are similar to the previous scenarios, this time Server 1 defines its policy using the FOAF vocabulary and imports the client's FOAF file, while Server 2 encodes user information in the vCard vocabulary. Thus, before the policies can be reconciled, FOAF and vCard need to be first

matched in order to produce bridge axioms. Once matched, PolVISor reconciles the two policies, resulting in the reconciled policy equivalent to that of Server 1. Figure 5 shows the process.

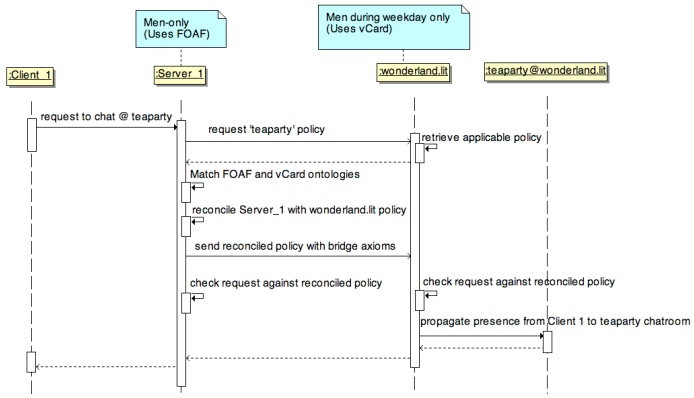

Figure 5: XMPP ontology matching.

The alignment between FOAF and vCard was used to dynamically produce OWL bridge axioms, which allow for reconciliation between policies using related, but differently named concepts. Thus, Server 2 can determine whether a client's foaf:nick has the required prefix, and subsequently is of the necessary gender, based on the "nickname" filed of his vCard. Although the scenario used a rather trivial example of matching, our design and implementation can support more complex alignments, as long as the matcher can first automatically align the ontologies.

VI. Conclusions

In this project, we have demonstrated that:

1. Policies authored in a restricted natural language format (SBVR Structured English) can be automatically converted to an executable formalism (BaseVISor rule language and OWL 2 RL) effectively.

2. Policies written in the ontology-based rule language provide an effective and flexible way to specify expressive policies that can be automatically enforced using ontology-based reasoning. The core ontologies used as the basis for domain-specific knowledge are grounded by our investigation of established security models.

3. Policies written in the ontology-based rule language can be effectively reconciled to allow for dynamic, policy-based information exchange between and an open set of XMPP servers.

4. While policy reconciliation typically requires the sharing of a common vocabulary, we have shown that effective ontology matching can be implemented to allow policy reconciliation across different (but similar) vocabularies.

Acknowledgment

This work was performed under US Army contract W91260-09-C-0037 "Security Policy Reconciliation".

References

[1] Semantics of Business Vocabulary and Business Rules v1.0. http://www.omg.org/spec/SBVR/1.0/ . May 2011.

[2] MDA Home Page. www.omg.org/mda/ . May 2011.

[3] MOF Home Page. www.omg.org/mof/ . May 2011.

[4] R. G. Ross. Basic RuleSpeak® Guidelines: Do's and Don'ts in Expressing Natural-Language Business Rules in English. http://www.rulespeak.com/en/Basic%20RuleSpeak%20Dos%20and%20Donts%20v2-2-5.pdf

[5] Business Rule Solutions, LLC. Home Page. http://www.brsolutions.com/ . May 2011M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[6] P. McDaniel and A. Prakash. Methods and Limitations of Security Policy Reconciliation. ACM Transactions on Information and System Security (TISSEC), Association for Computing Machinery, 9(3):259-291, 2006

[7] SBVR Visual Editor Home Page. http://sourceforge.net/projects/sbvrve/ . May 2011

[8] Sepiax-Web Page. http://www.sepiax.org/index.php?id=93 . May 2011

[9] M. De Tommasi, A. Corallo. SBEAVER: A Tool for Modeling Business Vocabularies and Business Rules. In Proc. 10th Int. Conf. on Knowledge-Based Intelligent Information and Engineering Systems (KES'06), LNCS Vol. 4253, 2006, 1083–1091

[10] MDT/SBVR Proposal Page. http://wiki.eclipse.org/MDT-SBVR-Proposal . May 2011.

[11] Eclipse Model Development Tools (MDT) Home Page. http://www.eclipse.org/modeling/mdt/ . May 2011.

[12] P. McNamara, Deontic Logic. *The Stanford Encyclopedia of Philosophy (Fall 2010 Edition)*, Edward N. Zalta (ed.). http://plato.stanford.edu/archives/fall2010/entries/logic-deontic/

[13] UCore Specification Page. https://www.ucore.gov/ . May 2011.

[14] B. Smith, L. Vizenor, J. Schoening. Universal Core Semantic Layer. Ontologies in the Intelligence Community Conference, 2007.

[15] Lee, R. Using New Standards to Develop IC Ontologies. In Proc. of the Fifth International Conference on Semantic Technologies for Intelligence, Defense, and Security. (STIDS'10). Fairfax, VA, USA, October 27-28, 2010.

[16] XMPP Standards Foundation. www.xmpp.org/ . May 2011.

[17] Openfire Home Page. http://www.igniterealtime.org/projects/openfire/

[18] Ignite Realtime Home Page. http://www.igniterealtime.org/ . May 2011.

[19] Spark Home Page. http://www.igniterealtime.org/projects/spark/index.jsp . May 2011.

[20] CWID 2010 UK Cross Domain Chat. Enclosure 1 to Cross Domain Chat Point Brief. July 2010.

[21] Boldon, James – Military Messaging and Secure Information Exchange Software Page. http://www.army-technology.com/contractors/navigation/boldonjames/ . May 2011.

[22] Isode Whitepaper: Using Security Labels to Control Message Flow in XMPP Services. http://www.isode.com/whitepapers/controlling-message-flow.html . May 2011.

[23] Common Information Sharing Standard for Information Security Marking: XML Implementation Implementation Guide. Office of the Director of National Intelligence Chief Information Officer. Release 2.0.3, February 2006.

[24] J. Euzenat, F. Scharffe, and A. Zimmermann, "Expressive alignment language and implementation," deliverable, Knowledge Web NoE, 2007. Available at http:// ftp//ftp.inrialpes.fr/pub/exmo/reports/kweb-2210.pdf..

[25] Kim, J. Luo, and M. Kang, "Security Ontology for Annotating Resources," Proceedings of 4th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE'05), Agia Napa, Cyprus, 2005.