Using SOA Provenance to Implement Norm Enforcement in *e*-Institutions

Javier Vázquez-Salceda and **Sergio Alvarez-Napagao**
Universitat Politècnica de Catalunya, Spain
{jvazquez,salvarez}@lsi.upc.edu

Abstract

In the last 10 years several approaches and technologies other than MAS (such as Web services and Grid computing) have emerged, with the support of the industry, providing their own solutions to distributed computation. As both Web services and Grid computing are based in the concept of service orientation, where all computation is split in independent, decoupled services, there is an opportunity for MAS researchers to test and extend their mechanisms and techniques in these emerging technologies. In this paper we describe a way to adapt the HARMONIA framework to be applied in highly regulated Web services and Grid computing scenarios. To do so we include a *provenance mechanism* as part of our norm enforcement mechanisms, which can be integrated into a SOA Governance workflow. We will show with an example how provenance allows the observation of both service interactions and (optionally) extra information about meaningful events in the system that cannot be observed in the interaction messages.

Introduction

With the growth of the Internet and the World Wide Web over the last fifteen years, previous metaphors for computation have been superseded by a new metaphor, of *computation as interaction*, where computing is not an action of a single computer but the result of a network of computers. Multi-Agent Systems (MAS) are one of the technologies that have emerged in this new metaphor. But they are not the only one. In the last 7 years other technologies such as Web services (World Wide Web Consortium (W3C) 2004) and Grid computing (Foster & Kesselman 1998) have emerged and matured, with the support of both the research community and the industry. These technologies are based in the concept of service-orientation (Erl 2004): a distributed system is comprised of units of service-oriented processing logic (the *services*) which hide their internal logic from the outside world and minimize dependencies among them. Recently some of these service-oriented technologies are converging into a single overarching framework, called Service-Oriented Architectures (SOA). Such framework is creating a collection of best practices principles[1]. But there are still deeper questions in the SOA community regarding the functioning of distributed systems using automated components. Many of these issues have been tackled in the research areas of Artificial Intelligence, Distributed Artificial Intelligence and, in particular, Multi Agent Systems research.

Thanks to the closeness between agent oriented and service-oriented approaches, some cross-fertilization between both technologies is feasible. The SOA community already has identified some potential to integrate agent research in SOA. For instance, Paurobally et. al have proposed to adapt and to refine Multi-Agent Systems research community results to facilitate the dynamic and adaptive negotiation between Semantic Web Services (Paurobally, Tamma, & Wooldridge 2005). Foster, Jennings and Kesselman already identified in (Foster, Jennings, & Kesselman 2004) the opportunity to have some joint research between the Grid and Agents communities.

In our view, there are also opportunities to apply both organizational and institutional approaches in SOA technologies in order to create a social layer on top of existing Web services and Grid platforms. To do so there are two main extensions to be done to SOA platforms:

- The introduction of additional semantics to the communication between services, in order to be able to check the actual behaviour of the actors in a distributed scenario from the intended behaviour.
- The introduction of higher-level behavioral control mechanisms, based in the extraction of some concepts such as commitments, obligations and violations, which can be derived thanks to some intentional stance extracted from the communication semantics.

There have been already some attempts for the first extension. An example is the work presented in (Willmott *et al.* 2005), where a connection between Agent Communication Languages and Web Service Inter-Communication is proposed, to then extend service communication with some FIPA performatives. The architecture we present in this paper uses this approach.

[1] Some of these principles are service abstraction (beyond what is described in the service contract, services hide logic from the outside world), service loose coupling (services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other) and service autonomy (services have control over the logic they encapsulate). See (Erl 2004) for more details and patterns in service-oriented design.

In the case of the second extension (introducing higher-level behavioral control mechanisms in SOA) it is necessary to have a language and a framework with which to model and manage the commitments. In this paper we present an approach that tackles this issue by defining a provenance-aware norm enforcement framework which combines agents and web services from an institutional approach, using substantive norms and landmarks.

SOA behaviour control and monitoring

Provenance

The aim of the IST-funded EU Provenance project was to conceive a computer-based representation of provenance in distributed service-oriented applications that allows users to perform useful analysis and reasoning. The *provenance* of a piece of data is the documentation of the process that produced the data. This documentation can be complete or partial (for instance, when the computation has not terminated yet); it can be accurate or inaccurate; it can present conflicting or consensual views of the actors involved; it can be detailed or not.

The Provenance architecture assumes that provenance is investigated in open, large-scale systems composed by services, seen as actors, that take inputs and produce outputs. In this abstract view, interactions between actors take place using messages. Actors may have internal states that change during the course of execution. An actor's state is not directly observable by other actors; to be seen by another actor, the state (or part of it) has to be communicated within a message sent by its owner actor. This architecture has formal foundations in the π-calculus (Milner 1999) and asynchronous distributed systems (Lynch 1996). The π-calculus is of interest in this context because of its approach to defining events that are internal to actors as hidden communications. This view also allows to formally define mappings with a) Grid applications, b) Web Services and c) Agent-Mediated Services and Applications.

Elements of the provenance architecture The provenance of a data item is represented in a computer system by a set of *p-assertions* made by the actors involved in the process that created it. A p-assertion is a specific piece of information documenting some step of the process made by an actor and pertains to the process. There are three kinds of p-assertions that capture an explicit description of the flow of data in a process. An *interaction p-assertion* is an assertion of the contents of a message by an actor that has sent or received that message. A *relationship p-assertion* is an assertion about an interaction, made by an actor that describes how the actor obtained output data or the whole message sent in that interaction by applying some function to input data or messages from other interactions. An *actor state p-assertion* is an assertion made by an actor about its internal state in the context of a specific interaction.

The long-term facility for storing the provenance representation of data items is the *provenance store*. The provenance store is used to manage and provide controlled access to the provenance representation of a specific data element.

Provenance life-cycle The *provenance life-cycle* is composed of four different phases. First, actors create p-assertions represent their involvement in a computation. After their creation, p-assertions are stored in a provenance store, with the intent they can be used to reconstitute the provenance of some data. After a data item has been computed, users or applications can query the provenance store. At the most basic level, the result of the query is the set of p-assertions pertaining to the process that produced the data. More advanced query facilities may return a representation derived from p-assertions that is of interest to the user. Finally the provenance store and its contents can be managed through a specific interface (subscription management, content relocation, etc).

Provenance awareness By transforming a MAS into a provenance-aware MAS, the resulting system gets the capability to produce at execution-time an explicit representation of the distributed processes that take place. Such representation can be then queried and analyzed in order to extract valuable information to validate, e.g., the basis of decisions taken in a given case, or to make an audit of the system over a period of time.

SOA Governance

SOA Governance[2] is an emergent concept in the SOA community used for activities related to exercising control over services (webMethods 2006). It is a form of electronic governance that has its focus on distributed services and composite architectures, more concretely on SOA scenarios.

In the last years many companies have started to switch to Service-Oriented Architectures for flexibility reasons and to adapt to technologies and practices under continuous growth and standardization. After adopting services as a kind of business asset, SOA Governance has appeared in the form of a methodology which affects the full life-cycle of the services in terms of specification, design, implementation, deployment, management, control, monitoring, maintenance, intercommunication, and redesign. Its aim is to give guidelines on how to establish shared policies, processes, architecture and policies across each layer of an organization.

SOA Governance tries to solve several issues, including: uncontrolled development of services that adapt usual process, usually leading to fragile services less robust than the previous implementation counterparts; lack of reusability, either because they are not designed with reusability in mind, or because they are not seen as valuable components in themselves; security compromise; and unexpected performance.

In summary, SOA Governance is intended to give the methodology and the tools needed to maintain the order in SOA environments. Some reports already try to identify how the community is doing at heading in this direction and which companies are on the good track and what do they lack of (Fulton 2008; Kenney & Plummer 2008).

[2] *SOA Governance* should not be confused with *E-Governance*. E-Governance can be defined as the use of Information and Communication Technology as a means to improve transparency, quality and efficiency of service delivery in the public administration.

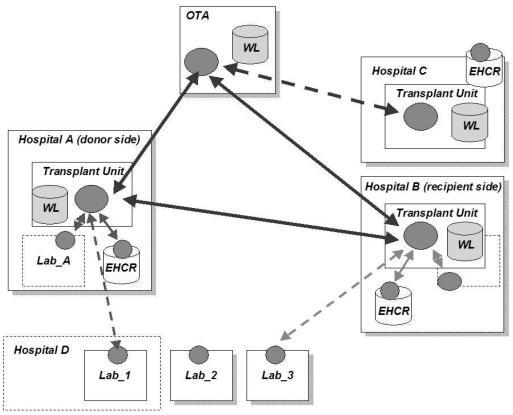

Figure 1: Actors in the OTMA system. Each medical unit is represented by an agent (circle in figure).

There are three steps that define SOA Governance management (webMethods 2006). Design-Time Governance deals with the definition and application of policies that will govern the design and implementation of Web services in the organization, prior to their deployment in the actual business environment. During Run-Time Governance, policies are defined and enforced in order to govern the deployment, execution, and use of the Web services. Eventually, web services are supposed to be redesigned and reimplemented in order to adapt to business evolving requirements. Change-Time Governance focuses on how the changes on the services affect the behaviour of a whole SOA environment.

The approach currently used in SOA Governance management is based on adding additional Web services in the SOA environment. The main components are:

- Registry: a central catalog for business services.
- Repository: a database of governance policies and metadata.
- Policy enforcement points: services responsible for the enactment of the policies.
- Rules engine: automatic system that manages the enforcement of the policies.
- Configuration environment: user interface for the configuration and definition of policies and governance workflows.

Use Case: The Organ Transplant Management Application

The Organ Transplant Management Application (OTMA) is an Agent-Mediated *e*-Institution for the distribution of organs and tissues for transplantation purposes. It extends CARREL (Vázquez-Salceda *et al.* 2003), the aim of which was to help speeding up the allocation process of solid organs for transplantation to improve graft survival rates. As opposed to CARREL, OTMA uses standard web service technology and is able to interact with provenance stores in order to keep track of the distributed execution of the allocation process for auditing purposes.

$Norm$	$OTM:N37$
$Condition$	OBLIGED($hospital$ DO $ensure\_compatibility(organ, recipient)$) BEFORE ($allocator$ DO $assign(organ, recipient)$))
$Violation\ condition$	NOT($done(ensure\_compatibility(organ, recipient))$ AND $done(assign(organ, recipient))$
$Sanction$	$inform(board,$ "NOT($done(ensure\_compatibility(organ, recipient))$ AND $done(assign(organ, recipient))$)")
$Repairs$	$\{stop\_assignation(organ);$ $assert($ NOT($done(ensure\_compatibility(organ, recipient))$ BEFORE $done(assign(organ, recipient)), p\_store$ $);$ $wait(asserted($ $ensure\_compatibility(organ, recipient)));$ $resume\_assignation(organ);\}$

Figure 2: Example of an OTMA norm

Figure 1 summarizes the different administrative domains (solid boxes) and units (dashed boxes) that are modeled in the OTMA system. Each of these interact with each other through agents (circles in the figure) that exchange information and requests through messages. In a transplant management scenario, one or more hospital units may be involved: the hospital transplant unit, one or several units that provide laboratory tests and the Electronic Healthcare Record (EHCR) subsystem which manages the health care records for each institution. The diagram also shows some of the data stores that are involved: apart from the patient records, these include stores for the transplant units and the Organ Transplant Authority (OTA) recipient waiting lists (WL). Hospitals that are the origin of a donation also keep records of the donations performed, while hospitals that are recipients of the donation may include such information in the recipient's patient record. The OTA has also its own records of each donation, stored case by case.

A Normative framework based in Norms and Landmarks

We use HARMON*IA*(Vázquez-Salceda 2004) as the basis for our normative framework, although the connection between the ideal states in the norms and the actual execution states of the system is done through the concept of landmarks, as in (Aldewereld *et al.* 2005).

In our normative framework we propose that enforcement of norms should not be made in terms of direct control of a central authority over the goals or actions that the agents may take, but through the detection of the *violation states* that agents may enter into and the definition of the *sanctions* that are related to the violations. With this approach we do not make strong assumptions about the agents' internal architecture, as the *e*-Institution only monitors the agent behaviour (that is, agents are seen as *black boxes*.). The enforcement of the norms in an *e*-Institution is achieved through a special kind of agents, the *Enforcement Agents*, which monitor the behaviour of the agents, detect violations and check the compliance of the sanctions.

Norms in organ transplant management

We use a language for substantive norms (Aldewereld *et al.* 2006) which is an evolution of the original norm language in HARMON*IA*. Its central element is the norm condition, based in deontic concepts (OBLIGED, PERMITTED, FORBIDDEN) which can be conditional (IF) and can include temporal operators (BEFORE, AFTER). The violation is a formula derived from the norm to express when a violation occurs. The sanction field is a set of actions which should be executed when a violation occurs (e.g. imposing a fine, expulsion of an agent), while the repairs field contains a set of actions to undo the negative effects of the violation. The language also included the specification of the detection mechanism, but in our provenance-based enforcement architecture this is no longer needed.

An example (extracted from organ and tissue allocation regulations) is presented in Figure 2. It expresses the obligation of the hospital to carry on the compatibility tests for a potential recipient of a given organ before assigning the organ to that recipient. The violation condition defines when the violation of that norm occurs. In this scenario, the sanctions field applies an indirect punishment mechanism (reputation) to the hospital, by informing about the incident to the board members of the transplant organization. The repair plan consists of stopping the assignation process, recording the incident in the provenance store (which acts as a log) and then wait for the compatibility test to be performed.

It is important to note that the combination of violation and sanction handling provides a flexible way to implement safety control of a medical system's behaviour (i.e., avoid the system to enter in a undesirable, illegal state because of a failure in one of the agents).

Control Landmarks

Landmarks (Aldewereld 2007) are often used with similar purposes in order to provide abstract specifications of organizational interaction in general. Landmarks are formalized as state descriptions, which are partially ordered in directed graphs to form landmark structures which are called *landmark patterns*.

In our case we extend the use of landmarks to represent highly relevant positive and negative states of the system (positive and negative landmarks) and the partial ordering between those states imposed by the regulations or practices. For instance, in the norm in Figure 2 we can identify two critical states as landmarks, the one where `ensure_compatibility` happens and the one where `assign` happens. The norm also imposes a partial ordering where the former should always happen before the latter.

Given the set of landmark patterns coming from the institution, agents may reason about the exact sequencing of actions or the protocol to use to pass from one landmark state to the other. This even allows an agent to create acceptable variations of a predefined protocol that are legal and that allow them to fulfill their interests or to cope with an unexpected situation not foreseen in the protocol. Given some landmarks, agents may even negotiate the protocol to use.

Landmarks can be used as checkpoints by the enforcing agents (e.g. whenever the assignation is done, it should be the case that previously the compatibility check was done). In short, norm enforcement can be done by checking that the system as a whole passes only through positive landmarks during its execution and in the proper order. In our system, landmarks are mapped into conjunctions of p-assertions, and landmark ordering is expressed in rules by means of the time stamps attached to each p-assertion. Figure 3 shows an example of how these p-assertions can be then used to detect a violation of the norm in Figure 2.

$Norm$	$OTM:N37$
$Violation$ $condition$	NOT($done(ensure\_compatibility(organ, recipient))$ AND $done(assign(organ, recipient))$
$Detection$ $condition$	(NOT($asserted(ensure\_compatibility(organ, recipient), t1)$ AND $asserted(assign(organ, recipient), t2)$) OR (($asserted(ensure\_compatibility(organ, recipient), t1)$ AND $asserted(assign(organ, recipient), t2)$ AND $(< t2 \;\; t1)$)

Figure 3: Example of a violation handling rule

An architecture proposal for norm enforcement in *e*-Institutions based in Provenance

In this section we introduce our proposal for a generic Provenance-based norm enforcement architecture. Although current version is mainly designed for Web service and Grid platforms, it can be easily adapted to be used also by agents in an agent platform. The global picture of this architecture is shown in Figure 4. When application agents enter for the first time in the *e*-institution, they can access the norms, the ontological definitions and the landmark definitions in the context manager module. Agents log the relevant events by creating p-assertions that are sent to the observer agent, which is the one that keeps the Provenance store that acts as a log for all the reported events. The observer agent sends some of those reported p-assertions to one or more Enforcement agents (each of those should have previously registered the list of p-assertions they need to be notified, according to the norms each of them has to enforce). Each enforcement agent combining the reported events in the p-assertions with the norms and landmarks that such agent is responsible to enforce. If a violation is detected, then the enforcement agent should execute the sanction and repair plans, as specified in the norms.

It should be noted that the *e*-Institution framework, HARMON*IA*, does not need to be modified for using Provenance. Provenance is only a different way to observe the state of a distributed system, which records and provides inputs (events and actions) that can be used for norm enforcement.

The following sections describe in detail each of the actors in our proposed architecture, focusing on their main roles and components.

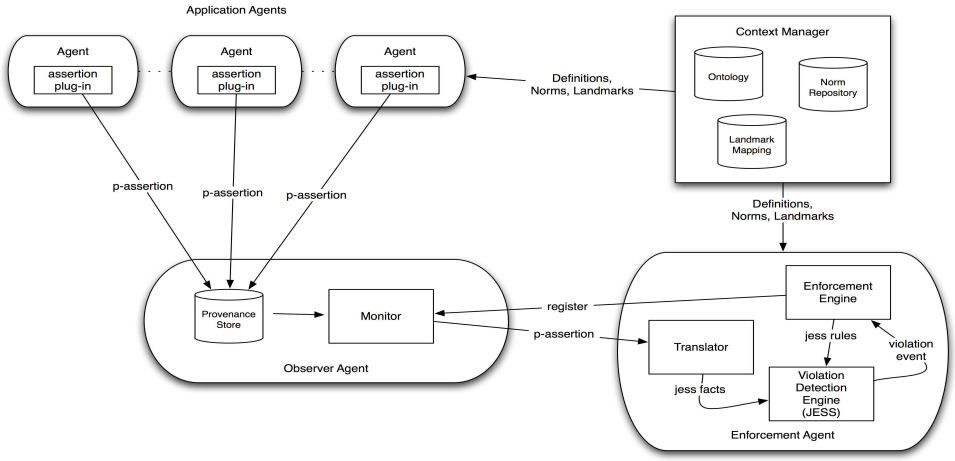

Figure 4: A generic Provenance-based norm enforcement architecture

Context Manager

In the approach taken for the architecture, every *e*-institution defines a normative context. This context gathers all the elements needed for understandability and interoperation between the agents belonging to a specific institution. The Context Manager is a registry responsible for the management of these elements and for providing to the agents any information related to the normative context.

An instance of this registry will represent a specific normative context, and will contain:

- a specific vocabulary defining the meaning of the terms used in the interactions between the agents of the institution,
- shared descriptions about processes and actions in the domain, and
- the norms that may affect the interactions between parties bound to the context.

To fulfill its responsibilities, the Context Manager has three main components, explained in the next subsections.

Ontology The Ontology is a repository which stores definitions of terms, as well as references to definitions, for the data models of the context. This ontology should define, for a given domain, terms such as objects and entities (e.g. patient, doctor, organ, kidney), predicates (e.g. compatible(organ, recipient)) and actions (e.g. assign(organ, recipient)). In our architecture the ontology plays an important role as it should fix the interpretation for all terms that appear in the norms to be enforced.

Norm Repository This module is responsible for storing and managing the norms of the *e*-institution. Each norm includes not only the deontic expression but also the violation condition, the sanction plan and the repairs plan.

Landmark Mapping This module is responsible for storing the mapping between landmarks and p-assertions. Such mappings can be used by both a) the application agents, to use the same p-assertion structure when reporting a relevant event that is listed as a landmark in the normative context of the *e*-institution; and b) the enforcement agents, that can use these mappings to translate the p-assertions they receive from the observer agent into landmarks.

Application Agent

The Application Agents are those agents that interact within each other inside the *e*-institution and its context. They have the same generic role as the agents in any typical multi-agent system and they do not necessarily have an active role in norm enforcement, but they should report all relevant events to the observer agent by creating p-assertions, which will be used by the enforcement agents to enforce the norms applying to the application agents' behaviour. P-assertion creation and reporting is handled by the p-assertion plug-in, a middleware component common to all Application Agents.

Before an Application Agent can start its activity within the *e*-institution, it has to retrieve the definitions, norms and landmarks of the context from the Context Manager. In this paper we make no assumption about the internal architecture of the agent and how this knowledge can be incorporated in the agent reasoning cycle. We also make no assumption about the exact technological platform in which it is implemented: it can be either a Web service, a Grid service or even a FIPA-compliant agent with a service wrapper that allows the agent to interact with the other actors in the architecture. Our only assumption is that the agents internal reasoning cycle has been modified to be able to report meaningful events (landmarks) through the Assertion Plug-in.

Assertion Plug-in This component is a middleware plug-in which manages the interaction between the application agents and the Provenance Store, ensuring a safe, reliable, and accurate recording of the events and landmarks generated by the agents execution. Whenever an agent wants to report the occurrence of a landmark:

1. The Assertion Plug-in translates this landmark into one or more p-assertions, by following the landmark mapping rules retrieved from the Context Manager.

2. The Assertion Plug-in sends the p-assertion(s) to the Observer Agent by using the Provenance Client API.

To avoid that p-assertions stop the execution of the agent or that some p-assertions get lost due to temporary unavailability communication problems between the Application Agent and the Observer Agent, the plug-in uses a p-assertion queue, which allows the p-assertion submission to be completely asynchronous and loosely coupled to the core of the agent, avoiding critically blocks in its execution.

Observer Agent

An Observer Agent has the responsibility to safely register and maintain the environmental events and state changes of the *e*-institution. The information gathered is then used in the norm enforcement, by providing selected pieces of information to the interested Enforcement Agents.

The gathering and the selection are critical processes. Some possible errors which depend on the Observer Agent and could compromise norm enforcement can take place, for example, if the events logged are not complete or reliable enough, or if the information provided to the Enforcement Agents doesn't match with their needs or arrives too late.

The gathering is handled by the Provenance Store which, along with the Assertion Plug-in, offers the proper recording functionalities. The Monitor acts as a link between this repository and the Enforcement Agents, offering registering and notification mechanisms. Both Observer Agent components are described in the subsections below.

Monitor The Provenance Store works only in a *push* way. The Enforcement Agents preferably need a real-time accurate representation of the *e*-institution, so the Observer Agent, as an actor, should behave in a *pull* way. That is why we have implemented the Monitor, layered on top of the Provenance Store. This component will keep an accurate real-time representation of the p-assertions being recorded in the Provenance Store.

Of course, this job should be handled efficiently, not only in time, but also in space, only keeping pointers to the p-assertions that are for some interest for the other agents. A registry is therefore incorporated to the Monitor, to which the Enforcement Agents subscribe with a list of mapped landmark patterns. While continuously reconstructing the real-time *picture* of the *e*-institution, the Monitor will just query those p-assertions which match with the patterns of the Enforcement Agents registered. As soon as a p-assertion has appeared in the Provenance Store that matches a registration pattern of an Enforcement Agent, this p-assertion is sent to the registrant.

Provenance Store The Provenance Store is usually an independent service, but we consider it as part of the Observer Agent, as these will be the only actors of the *e*-institution which will make use of them. As a repository of raw p-assertions, it will only receive one kind of input, provided by the Assertion Plug-ins of the Application Agents. As well, it will only generate one kind of output, in this case the result of the queries made by the Monitor, as sets of p-assertions.

Enforcement Agent

The Enforcement Agents are responsible for the fulfillment of a subset of the norms of the context in the *e*-institution. This requires them to have a complete knowledge of the context, by retrieving the descriptions and the norms from the Context Manager, as well as a complete knowledge of all the events in the system related to the norms they have to enforce. These enforcement is then guaranteed by a) firstly detecting the violations, and then b) applying the corresponding sanctions.

In order to generate the knowledge about the events, these agents take profit of the Observer Agent by registering the set of landmarks they are supposed to look after. Once registered, they will be properly notified in the form of p-assertions. Therefore, there is no need of a direct communication between an Enforcement Agent and the Application Agents. The Translator converts these p-assertions into a format understandable by the Enforcement Agent. Another component is needed for detecting the violations. In our case we are using a Jess engine, which matches the events, in the form of Jess facts, and the norms, in the forms of Jess rules. The Enforcement Engine is responsible for registering to the Observer Agents and applying sanctions. A further explanation of how this component works is also included below.

Translator The Observer Agent sends p-assertions to the Enforcement Agent when they are of any interest. However, the Violation Detection Engine is an instance of a Jess engine. The Translator is a simple component which parses these p-assertions and generates Jess facts.

```
(defrule OTM-RULES-MODULE::assertconfirmassignment
  (MAIN::Element (LocalName "opencontent")
    (ElementID ?content))
  (MAIN::Element (LocalName "timestamp")(Text ?timestamp)
    (ParentID ?content))
  (MAIN::Element (LocalName "confirmassignment")
    (ElementID ?confirmassignment)(ParentID ?content))
  (MAIN::Element (LocalName "organ")(Text ?organ)
    (ParentID ?confirmassignment))
  (MAIN::Element (LocalName "pid")(Text ?pid)
    (ParentID ?confirmassignment))
  (not (OTM-RULES-MODULE::confirmassignment
    (ElementID ?confirmassignment)(timestamp ?timestamp)
    (organ ?organ)(pid ?pid)))
=>
  (assert (OTM-RULES-MODULE::confirmassignment
    (ElementID ?confirmassignment) (timestamp ?timestamp)
    (organ ?organ)(pid ?pid))))
```

Figure 5: An example of translation rule from p-assertion to Jess $asserted$ fact

The Translator obtains the translation rules from the Context Manager. In Figure 5 we show one example of a rule that obtains a Jess assertion of an organ assignment, taking an organ assignment p-assertion as input. This rule parses the XML formatted p-assertion, keeping only the relevant data for the system and generating an asserted fact, which will be added to the Jess engine. In this case, the rule is involved in the moment that the doctor of a hospital accepts

the organ offer and therefore confirms the assignment proposed by the OTA. According to the medical protocol being followed, the relevant pieces of data in this step are the exact moment of the assignment, the recipient patient identifier, and the organ. They are retrieved from the XML p-assertion and written in a Jess fact.

When an agent records a p-assertion indicating the confirmation of an assignment, it includes content compliant with the OTMA XML schema. On the left side, this rule matches one by one the elements contained inside the *open-content* element: the exact moment of the action, the name of the event (*confirmAssignment*), and inside the *confirmAssignment* element, the organ being proposed for reception and the ID of the recipient. After the matching, the left side of the rule checks that there was no assertion made yet for the same event. On the right side, the rule asserts the event *confirmAssignment* into the base of facts.

He have implemented an automatic translator of rules, capable of parsing an schema and generating one rule per each kind of event the content of the p-assertion might contain, which right now we assume is once per each XML element defined. It will be improved in future releases.

Violation Detection Engine Once the Enforcement Engine has received the norms from the Context Manager, it creates a set of Jess rules out of them and sends them to the Violation Detection Engine. This component is, in fact, an instance of a Jess engine which will execute these rules with the facts provided by the Translator. Whenever a violation is detected, the Enforcement Engine is conveniently informed.

Enforcement Engine The Enforcement Engine is the component of the Enforcement Agent that takes decisions and plans actions whenever a violation is raised. In order to interact with the Violation Detection Engine, this component needs to provide Jess rules for each norm.

The violation for the norm *N37* has to be raised whenever, in the confirmation of an assignment, this assignment has been made before having checked for compatibility. This might happen when the assignment is done but the compatibility is never ensured. But also when both things are done, but in the wrong order. This second case is the one depicted in Figure 6. The rule shown in the figure takes as input two facts: the fact generated (using the translation rule shown in Figure 5) when the hospital confirmed the assignment of the offered organ to the doctor, and the fact generated when the organ was tested for compatibility. The third condition of the rule, `(< t2 t1)`, will become true if the assignment has been done before the compatibility test. Whenever the rule gets executed, a violation fact for the norm *N37* will be added to the Jess engine and the Enforcement Agent will, at some point, take measures to repair the violation.

The Enforcement Agent will act accordingly to the type of measures needed. If the sanction or the repair measures require that a specific Application Agent executes a certain action, that agent will be informed of that. On the other hand, the sanction or the repair measures that involve the institution as itself will be carried into effect by the Enforcement Agent.

When an Enforcement Agent is initiated, the ontological

```
(defrule OTM-RULES-MODULE::eventOTM_N37_2
(OTM-RULES-MODULE::ensure_compatibility (organ ?organ)
  (recipientID ?recipientID)(timestamp ?t1))
(OTM-RULES-MODULE::assign (organ ?organ)
  (recipientID ?recipientID)(timestamp ?t2))
(< t2 t1)
=>
(assert (OTM-RULES-MODULE::violation (norm OTM_N37)
  (organ ?organ)(recipientID ?recipientID)))
```

Figure 6: An example of violation detection rule in Jess

definitions and the norms of the context are stored in its Enforcement Engine. This component is also the responsible for registering to the Monitor.

For the norm example shown in Figure 2, all the measures should be executed by the Enforcement Agents, as they are all institutional.

Related work

AMELI (Esteva 2003) is a toolkit for the specification and verification of agent mediated *e*-institutions that based on a dialogical framework. In this framework, all observable activities from the agents are seen as messages in the context of a *scene*. In AMELI all norms are regimented through the specification of a pre-defined protocol, guaranteeing norm-compliance of agents by restricting the set of possible actions to the ones defined in the protocol.

In (Aldewereld *et al.* 2006; Aldewereld 2007) there is a first exploration of substantive norms already applied to AMELI. The main difference in our approach is that we can also include internal information from agents which is not part of any interactions. On the other hand, the formalism defined in (Cliffe, De Vos, & Padget 2008) only considers messages as observable events.

(Ashri *et al.* 2006) introduces integration of *e*-Institutions in Grid environments by extending the GRIA framework, which is based on basic web services. Our solution gives more flexibility to the behaviour of the services, as norms are substantive and not rigidly regulated.

It is important to note that the provenance mechanism used here is an implementation of an open architecture (Groth *et al.* 2006) that ensures interoperability in heterogeneous systems without compromising security or scalability. Other provenance mechanisms are mainly based on a middleware layers which only capture interactions. This kind of provenance mechanisms can be used in less regulated environments but bring little extra power to the existing mechanisms in agent-mediated *e*-Institutions.

Academic research on SOA Governance is still not abundant, but there are already some interesting proposals of models (Derler & Weinreich 2007), methodologies (Zhou *et al.* 2007; Papazoglou 2006) and frameworks (Kajko-Mattsson, Lewis, & Smith 2007). In our paper we present a novel approach to the topic based on flexible normative enforcement via landmark monitoring.

Conclusions

The fact that the SOA business community is concerned about how to define and manage policies for the definition, deployment and change of Web services is a clear sign that organizations need to translate and adapt their own business processes and methodologies of work in their SOA environments. Electronic Institutions respond to the need of regulation in MAS that have to be bound to certain norms that apply in the context of an institution. They provide a theoretical solution that could match many of the needs of SOA Governance as, once the policies are defined, an *e*-Institution framework could take care of their enforcement.

However, SOA Governance does not focus on MAS. With our architecture proposal we aim at bridging this gap by combining Web services and agents inside a normative framework derived from HARMON*IA*, and deployed in heterogeneous (MAS, Web services and/or Grid) platforms.

As next steps we will define a mapping between the operational representation of norms and: 1) orchestration languages, which would allow us to better integrate our proposed architecture into business processes, as well as 2) choreography languages, which would give us the possibility of extending the uses of the interaction provenance recording. By defining a mapping, norms could be instantiated in languages like WS-BPEL or WS-CDL, which could be imported directly by web service workflow engines which are currently widely used.

Acknowledgment

This work has been funded by IST-2002-511085 PROVENANCE and IST-2006-034418 CONTRACT projects. Javier Vázquez-Salceda's work has been also partially funded by the "Ramón y Cajal" program of the Spanish Ministry of Education and Science.

References

Aldewereld, H.; Grossi, D.; Vázquez-Salceda, J.; and Dignum, F. 2005. Designing Normative Behaviour by the Use of Landmarks. *Proc. of AAMAS-05 Int. WS on Agents, Norms and Institutions for Regulated MAS*.

Aldewereld, H.; Dignum, F.; García-Camino, A.; Noriega, P.; Rodríguez-Aguilar, J. A.; and Sierra, C. 2006. Operationalisation of norms for usage in electronic institutions. In *AAMAS'06: Proc. of the 5th Int. Joint Conf. on Autonomous Agents and MAS*, 223–225. NY, USA: ACM.

Aldewereld, H. 2007. Autonomy vs. conformity : an institutional perspective on norms and protocols. PhD Thesis.

Ashri, R.; Payne, T. R.; Luck, M.; Surridge, M.; Sierra, C.; Aguilar, J. A. R.; and Noriega, P. 2006. Using Electronic Institutions to secure Grid environments. In *Tenth Int. WS CIA 2006 on Cooperative Information Agents*, 461–475.

Cliffe, O.; De Vos, M.; and Padget, J. 2008. Embedding Landmarks and Scenes in a Computational Model of Institutions. In *Coordination, Organizations, Institutions, and Norms in Agent Systems III*. 41–57.

Derler, P., and Weinreich, R. 2007. Models and Tools for SOA governance.

Erl, T. 2004. *Service-Oriented Architecture*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Esteva, M. 2003. Electronic Institutions: from specification to development. *PhD Thesis, UPC*.

Foster, I., and Kesselman, C. 1998. *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Foster, I.; Jennings, N. R.; and Kesselman, C. 2004. Brain meets brawn: why Grid and agents need each other. In *3rd Int. Conf. on Autonomous Agents and MAS*, 8–15.

Fulton, L. 2008. The Forrester Wave: SOA Service Life-Cycle Management, Q1 2008.

Groth, P.; Jiang, S.; Miles, S.; Munroe, S.; Tan, V.; Tsasakou, S.; and Moreau, L. 2006. An architecture for provenance systems.

Kajko-Mattsson, M.; Lewis, G. A.; and Smith, D. B. 2007. A Framework for Roles for Development, Evolution and Maintenance of SOA-Based Systems. In *ICSEW'07: Proc. of the 29th Int. Conf. on Software Engineering WS*, 117. Washington, DC, USA: IEEE Computer Society.

Kenney, L. F., and Plummer, D. C. 2008. Magic Quadrant for Integrated SOA Governance Technology Sets, 2007. *Gartner RAS Core Research Note G00153858*.

Lynch, N. 1996. Distributed Algorithms. *The Morgan Kaufmann Series in Data Management Systems*.

Milner, R. 1999. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press.

Papazoglou, M. P. 2006. Service-oriented design and development methodology. *Int. Journal of Web Engineering and Technology* 2:412–442(31).

Paurobally, S.; Tamma, V.; and Wooldridge, M. 2005. Cooperation and agreement between semantic web services. *W3C WS on Frameworks for Semantics in Web Services*.

Vázquez-Salceda, J.; Cortés, U.; Padget, J.; López-Navidad, A.; and Caballero, F. 2003. The organ allocation process: a natural extension of the carrel agent-mediated electronic institution. *AI Commun.* 16(3):153–165.

Vázquez-Salceda, J. 2004. *The role of norms and electronic institutions in multi-agent systems. The HARMONIA framework. PhD Thesis*. Whitestein Series in Software Agent Technologies. Birkhäuser.

webMethods. 2006. SOA Governance: Enabling Sustainable Success with SOA. http://www1.webmethods.com/PDF/whitepapers/.

Willmott, S.; Pea, F. O. F.; Merida-Campos, C.; Constantinescu, I.; Dale, J.; and Cabanillas, D. 2005. Adapting agent communication languages for semantic web service inter-communication. In *WI'05: Proc. of the 2005 IEEE/WIC/ACM Int. Conf. on Web Intelligence*, 405–408. Washington, DC, USA: IEEE Computer Society.

World Wide Web Consortium (W3C). 2004. Web services architecture. http://www.w3.org/TR/ws-arch/.

Zhou, Y. C.; Liu, X. P.; Kahan, E.; Wang, X. N.; Xue, L.; and Zhou, K. X. 2007. Context aware service policy orchestration. *icws* 0:936–943.