



Departament de Llenguatges
i Sistemes Informàtics

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Computational Mechanisms for Norm Enforcement in Service-Oriented Architecture

DEA+PT

Autor: Sergio Alvarez Napagao

Director: Javier Vázquez Salceda



Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA

institution

noun

1. a society or organization founded for a religious, educational, social, or similar purpose : *a certificate from a professional institution.*
 - an organization providing residential care for people with special needs : *an institution for the severely handicapped.*
 - an established official organization having an important role in the life of a country, such as a bank, church, or legislature : *the institutions of democratic government.*
 - a large company or other organization involved in financial trading : *the interest rate financial institutions charge one another.*
2. an established law, practice, or custom : *the institution of marriage.*
 - informal a well-established and familiar person, custom, or object : *he soon became something of a national institution.*
3. the action of instituting something : *a delay in the institution of proceedings.*

New Oxford American Dictionary, 2nd Edition, 2005.

enforce

verb [trans.]

compel observance of or compliance with (a law, rule, or obligation).

- cause (something) to happen by necessity or force : *there is no outside agency to enforce cooperation between the players* | [as adj.] **(enforced)** *a period of enforced idleness.*

New Oxford American Dictionary, 2nd Edition, 2005.

Contents

Contents	v
List of Figures	vii
1 Introduction	1
1.1 Computation as interaction	1
1.2 Service-Oriented Architectures	2
1.3 The need for control: SOA Governance	3
1.3.1 Governance 3, 1.3.2 Corporate governance 4, 1.3.3 IT Governance 4, 1.3.4 SOA Governance 5	
1.4 Electronic Institutions for SOA Governance	5
1.4.1 Norms 6, 1.4.2 Institutions 11	
1.5 Objectives	13
1.6 Use case: the OTM application	14
1.6.1 Distributed Transplant Management: the OTM application 14	
1.7 Structure of this document	15
2 Behaviour Monitoring in SOA: Provenance	17
2.1 Provenance	18
2.1.1 Related Work to Provenance 18, 2.1.2 Provenance in Service Oriented Architectures 19, 2.1.3 Elements of the provenance architecture 19	
2.2 Applying Provenance in a distributed SOA-based system	20
2.2.1 Provenance Handling in the OTM Application 21, 2.2.2 Adapting the OTMA system for Provenance 21, 2.2.3 Process Documentation in Provenance Aware HC-MAS 23, 2.2.4 Protecting Privacy in the Provenance-aware Application 27	
2.3 Conclusions	29
3 Behaviour Enforcement in SOA: From SLA to Electronic Contractual Institutions	31
3.1 SLA in SOA	31
3.2 An intermediate step: Contractual Institutions	33
3.3 Contracting language	34
3.3.1 Layers/Elements of the Contracting Language 34, 3.3.2 Contract representation 36, 3.3.3 Contracting Messages and Protocols 43	
3.4 Contractual Service Middleware	44

	3.4.1 Contracting Environment 46, 3.4.2 Administrative contract parties 46, 3.4.3 Internal architecture 47, 3.4.4 Supporting Systems 49, 3.4.5 Example of Contract Management and Execution 49	
3.5	Conclusions	52
4	Thesis Proposal	55
4.1	HARMONIA	55
	4.1.1 The Abstract Level: statutes, objectives, values and norms 56, 4.1.2 The Concrete Level: from abstract norms to concrete norms 57, 4.1.3 The Rule Level: translating norms into rules 57, 4.1.4 The Procedure level 58, 4.1.5 Policies 59, 4.1.6 Role hierarchy 59, 4.1.7 Ontologies 59	
4.2	Arquitecture of a SOA governance system	60
	4.2.1 SOA governance requirements 60, 4.2.2 Technologies for SOA gover- nance 63	
4.3	A Contractual Institution- and Provenance-based Norm Enforcement Mechanism	68
	4.3.1 A Normative framework based on Norms and Landmarks 68, 4.3.2 An architecture proposal for norm enforcement in SOA 70, 4.3.3 Related work to our proposal 76	
4.4	Conclusions	76
5	Working Plan	79
	Bibliography	81
	Publications	91
	Glossary	93

List of Figures

1.1	The OTMA system. Each medical unit is represented by an agent (circle in figure) which manages interactions with other units and with the EHCR subsystem.	15
2.1	Example scenario: Interactions of the OTMA agents involved in a donation decision.	22
2.2	Example scenario: Directed acyclic graph showing the provenance of the donation decision.	23
2.3	Process documentation in strongly connected processes. P-assertions of strongly connected processes are linked together by the p-assertions related to the interaction connecting the two processes.	25
2.4	Linking of process documentations in weakly connected healthcare processes. P-assertions of weakly connected healthcare processes are linked together by the p-assertions of a higher level service.	27
3.1	General view of the Contracting Framework	35
3.2	Root elements of the Contract representation	36
3.3	Parties element in contract	37
3.4	Role enactment list element in contract	37
3.5	Representation of the Group list element	37
3.6	Representation of the world model element	38
3.7	Representation of the clause element	39
3.8	Representation of the deontic statement element	40
3.9	Simple Contract Create protocol	45
3.10	The contracting environment in the example <i>buyMusic</i> scenario	46
3.11	Agent Service Middleware Architecture	48
3.12	The workflow derived from the <i>buyMusicContract</i> clauses. During the <i>download</i> an exception may occur which triggers the activation of violation clause <i>dc5</i> and the dotted path of the workflow is followed.	51
3.13	The <i>musicBuyer</i> and <i>musicSeller</i> interactions and internal states regarding the contract clauses for the <i>buyDownloadRights</i> action (a), <i>pay</i> action (b) and the <i>pay</i> clause violation (c) respectively.	52
4.1	Multi-level architecture for Virtual Organizations.	56
4.2	Key components of a SOA governance system	60

4.3	Example of an OTMA norm	69
4.4	Example of a violation handling rule	70
4.5	A generic Provenance-based norm enforcement architecture	71
4.6	An example of translation rule from p-assertion to Jess <i>asserted</i> fact	74
4.7	An example of violation detection rule in Jess	75

Introduction

This Chapter contains definitions about fundamental concepts and the use case used through this document, as well as the objectives of this PhD proposal.

In Section 1.1, we treat the concept of *computation as interaction*, which will lead to *service-orientation*, explained in Section 1.2. From the SOA issues, we will focus on the need for control in the process of SOA adoption and thus in *SOA governance*, found in Section 1.3. We will then explain in Section 1.4 how the concept of *Electronic Institutions* can be useful to support and extend SOA governance.

After all these concepts are summarized, we will present the *objectives* of this PhD proposal in Section 1.5. A short description of the *use case* used later in several parts of the document can be found in Section 1.6. Finally, the *structure* for the rest of this document is shown in Section 1.7.

1.1 COMPUTATION AS INTERACTION

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 one, called *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 in the last years other technologies such as Web services [BHM⁺04] and Grid computing [KF04] have emerged and matured, with the support of both the research community and the industry. These technologies are based on the concept of service-orientation. Recently some of these service-oriented technologies are converging into a single overarching framework, called Service-Oriented Architectures (SOA) (see Section 1.2 for more details about SOA).

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.

Although MAS were not originally developed as part of the Service-Oriented Architectures, parallels can be found between both approaches. For instance, some agent

applications use agents to distribute computation by offering services to other agents in the community.

Thanks to the closeness between agent oriented and service-oriented approaches, some cross-fertilization between both technologies is feasible. The SOA community has already 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 [PTW05]. Foster, Jennings and Kesselman already identified in [FJK04] 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 behavior of the actors in a distributed scenario from the intended behavior.
- The introduction of higher-level behavioral control mechanisms, based on 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 [WPMC⁺05], where a connection between Agent Communication Languages and Web Service Inter-Communication is proposed, to then extend service communication with some FIPA performatives.

In the case of the second extension, it is necessary to have a language and a framework with which to model and manage the commitments.

In particular, Multi-Agent Systems technologies are important as a foundation for the commitments, obligations and other relationships that are likely to be required for service-based interactions, in order to function effectively in open environments.

1.2 SERVICE-ORIENTED ARCHITECTURES

Service-orientation [ANOB⁺07] presents an ideal vision of a world in which resources are cleanly partitioned and consistently represented. When applied to IT architecture, service-orientation establishes a universal model in which automation logic and even business logic conform to this vision. This model applies equally to a task, a solution, an enterprise, a community, and beyond.

When applied to automation technology, service-orientation represents a distinct approach for analysis, design, and development by introducing principles that govern aspects of communication, architecture, and implementation of processing logic. The more a solution is comprised of units of service-oriented processing logic, the more it becomes a service-oriented solution. The following are principles of service-orientation [Erl04]:

- *Service reusability*: logic is divided into services with the intention of promoting reuse.
- *Service contract*: services adhere to a communications agreement, as defined collectively by one or more service description documents.

- *Service loose coupling*: services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.
- *Service abstraction*: beyond what is described in the service contract, services hide logic from the outside world.
- *Service composability*: collections of services can be coordinated and assembled to form composite services.
- *Service autonomy*: services have control over the logic they encapsulate.
- *Service statelessness*: services minimize retaining information specific to an activity.
- *Service discoverability*: services are designed to be outwardly descriptive so that they can be found and accessed via available discovery mechanisms.

A *Service Oriented Architecture* (SOA) represents a collection of best practices, principles and patterns in service-oriented design. The main drivers for SOA adoption are that it links services and promotes their reuse and decoupling.

There are many service oriented architectures, frameworks and initiatives available nowadays. The OASIS Service Oriented Architecture Reference Model (SOA-RM) [MLM⁺06] is a reference model for core Service Oriented concepts developed to guide and foster the creation of more specific, service-oriented architectures. The W3C Web Services Architecture (W3C WSA) [BHM⁺04] identifies the functional components of a Web Service architecture and defines the relationships among those components to effect the desired properties of the overall architecture. The Open Gateway Service initiative [CGD⁺99], Grid [KF04] and JINI [Arn99] architectures are all service oriented architectures, proving that SOA is widely accepted as a paradigm.

1.3 THE NEED FOR CONTROL: SOA GOVERNANCE

One main concern about SOA is that it supposes a paradigm shift for the industry [IHJ⁺07]. SOA adoption is complex in the sense that, in each particular case, the possible successful implementations are too many and the outcomes are too unpredictable to make of it an straightforward process. On the contrary, this is a process that needs control to some extent. This is an issue that is being currently covered by *SOA governance*.

SOA governance is a relatively new but growing and evolving concept that derives from the more generic of governance. It is a compendium of practices, methodologies, architectures, frameworks and technologies that seek a better control and management over SOAs, in their whole lifecycle.

In the following subsections we describe the roots of SOA governance. Its methodological aspects, which will help us later in identifying matching points between SOA governance and Electronic Institutions, are explained in Section 4.2.

1.3.1 Governance

According to the dictionary, governance can be defined as *the action of conducting, influencing, or regulating the policy, actions, and affairs of a state, a organization, or a group a people*. This can be done by constituting laws, rules, standards, or principles [SW03]. Governance, in practice, is defined by Finkelstein [Fin95] as systems of rule at all levels of human activity in which the pursuit of goals through the exercise of control has repercussions.

Rosenau and Czempiel [RC92] make a distinction between governance and government. Both refer to purposive behavior, to goal-oriented activities, and to systems of rule. But government suggests activities that are backed by formal authority, by police owners

to insure the implementation of according constituted policies, whereas governance refers to activities backed by shared goals that may or may not derive from legal and formally prescribed responsibilities and that do not necessarily rely on police owners to overcome defiance and attain compliance.

Governance, in other words, is a more encompassing phenomenon than government. It embraces governmental institutions, but it also subsumes informal, non-governmental mechanisms whereby those persons and organizations within its scope move ahead, satisfy their needs, and fulfill their wants. Governance is a system of rule that works only if it is accepted by the majority, whereas governments can function even in the face of widespread opposition to their policies.

1.3.2 Corporate governance

Corporate governance is the subset of governance that involves corporations. That is, the set of laws, rules, institutions, and policies which affect the way a corporation is controlled and managed [MM04]. The actors involved in corporate governance are the shareholders, the board of directors, the management and other stakeholders¹, although in most cases it is a delegated responsibility on the managers and the board, bounded by public legislations and regulations.

For Colley et al. [CDSL03], corporate governance is a concept of growing interest, being critical nowadays in modern economies. This is due to a continuous criticism over the years about the way corporate boards work, and the constant pressure over the fulfillment of the responsibilities they have been elected for.

Colley et al. claim that this is the main collective problem of business as of today, and present a template for corporate governance analysis from several perspectives:

- *The legal issues*: what does the law require?
- *The ethics*: how does the organization define and fulfill its obligations to its constituencies or stakeholders in view of conflicting interests?
- *Effectiveness*: how does the board ensure that they and its management make effective decisions in an efficient and timely manner?
- *The board's relationships*: how does the board maintain effective relationships with their constituencies, particularly shareholders and management?
- *The group dynamics*: how well does the board function as a group or team?

1.3.3 IT Governance

The definitions of governance and corporate governance seem to be clear and more or less uniform. This is not the case with IT governance. However, Simonsson and Johnson [SJ06] analyzed over 60 articles in order to extract an approximate shared definition:

IT Governance are the tactics and the strategy (decision-making process) made and carried out in order to monitor, decide and understand (scope) upon the IT goals, technologies, human resources and processes (domain) of an organization.

In most of the organizations, every IT resource and process has some level of governance associated with it in the form of policies, rules, and controls. However, according to Webb et al. [WPR06], companies with a well designed IT governance earn a higher return on their assets. In fact, it is widely accepted that it directly influences the benefits

¹Customers, employees, suppliers or the community at large are examples of stakeholders.

generated by organizational IT investments [Wei04]. However, despite being seen as an essential component in the overall corporate governance structure, IT governance is right now the weakest link in the chain [Tri04].

1.3.4 SOA Governance

Whereas IT Governance deals with hardware and software in general terms, SOA governance is concentrated only on the Service-Oriented Architectures subset. Moreover, *SOA Governance* is an emergent concept in the SOA community used for activities related to exercising control over services [web06]. It is a form of electronic governance that has its focus on distributed services and composite architectures, more concretely on SOA scenarios, which may be under the control of different ownership domains.

Initially, the concept of SOA governance was applied narrowly to the development and use of Web services, for example, validating the conformance of a Web service with specific standards or managing Web services in the SOA run-time environment. 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:

- Fragile and delicate SOA implementations.
- Services that cannot easily be reused because they are unknown to developers or because they were not designed with reuse in mind.
- Lack of trust and confidence in services as enterprise assets, which results in a *build it myself* mentality, further compounding the lack of reuse with redundancy and unnecessary duplication of functionality.
- Security breaches that cannot easily be traced.
- Unpredictable performance.

In summary, SOA Governance is intended to give the methodology and the tools needed to maintain the order in SOA environments. Some periodic reports are already trying to identify how the community is doing at heading in this direction and which companies are on the good track and what they lack [FHD08, KP08].

1.4 ELECTRONIC INSTITUTIONS FOR SOA GOVERNANCE

The fact that the SOA business community is concerned [KP08] with 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 those MAS in which the agents 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.

Institutions are usually defined in a formal way by the use of *norms* and their effect in the actions performed by their members. Therefore, in Section 1.4.1 we will introduce the concept of norms, and in Section 1.4.2 we will describe the concept of institution at a human and at an electronic level.

1.4.1 Norms

A *norm* can be defined from several perspectives [VS04]. It can be seen as a rule or standard of behavior shared by members of a social group, as an authoritative rule or standard by which something is judged and, on that basis, approved or disapproved, as standards of right and wrong, beauty and ugliness, and truth and falsehood, or even as a model of what should exist or be followed, or an average of what currently does exist in some context.

In [VS04], Vázquez-Salceda makes a distinction of the treatment of norms in different contexts: *Social Theory*, *Legal Theory*, and *Computer Science*.

Norms in Social Theory

Regulation modelling was already a concern in ancient Greece and Rome, for example in Plato's Republic [Pla68] and Aristotle's Ethics [Ari57]. Classical approaches studied a society as a closed system where all the relations among individuals can be defined and established.

Modern approaches are focused on an open systems view where the environment or context has an important influence as it *constraints, shapes, penetrates* and *renews* the society or organization [Sco98]. In this new scenario, problems such as consensus or limited trust may affect the interactions of the individuals when trying to achieve coordination or cooperation. Human societies have successfully coped with similar problems of *social order*, mainly by the development of *norms* and *conventions* as behavior specifications to be followed by individuals.

Institutional Theory [NNS90][Sco01] appeared as part of the study of the role of norms in open systems. Its main thesis is that, in some societies, norms are supported by social institutions, which enforce the fulfillment of the norms by the members of the society. Institutional theory presents two approaches of research: the study of the dynamics of a regulatory framework, that is, how norms emerge and get enforced; and the effects of regulatory frameworks in the dynamics of a social system, and how the behavior of its individuals is adapted.

North [NNS90] studied the effect of institutions, in the sense of normative systems, on the behavior of human organizations, focusing on their performance. According to him, *institutional constraints* reduce the cost of human interactions, by ensuring trust between parties and giving shape to their choices. When there are institutional constraints, individuals are able to behave, and expect others to behave, according to the norms.

Scott [Sco01] defines a framework to describe the interactions between institutions and organizations. In this framework, an institution is composed by the regulative aspects, the normative ones, and the cultural-cognitive ones. In Scott's view, there is a distinction to be made between norms, which are related to moral aspects and describe *regulative aspects* of individual behavior, and rules, which are related to coercive aspects and describe *prescriptive, evaluative, and obligatory aspects* of social life.

Moreover, for Scott, roles are conceptions of appropriate goals and activities for particular individuals or specified social positions, creating, rather than anticipations or predictions, prescriptions, in the form of normative expectations, of how the actors fulfilling a

role are supposed to behave. These roles are directly related to those norms that are not applicable to all members of a society, but to selected types of actors.

Norms in Legal Theory

Although primitive cultures had no written laws, they shared morals and conventions that were followed by the individuals. These *informal norms* evolved into explicit written laws developed by subsequent cultures. Two examples of these *formal norms* are *Roman Law* and *Common Law*.

Roman Law emerged after some attempts of expressing morals and conventions, e.g. the code of Hammurabi, and is the base for the legal systems of most of the european countries that were part of the Roman Empire, the ones that were linked to them by monarchic marriages, as well as their colonies. In most of these countries the normative system is hierarchical: *constitution, laws, and regulations*.²

Roman Law defines two kinds of constraint sets:

- Normatives or Laws: sets of *norms* that define WHAT can be done by WHO and WHEN. A constitution is the highest law in a state.
- Regulations: sets of *rules* that expand a given normative defining HOW it may be applied.

In legal theory, norms are always expressed in natural language. That makes them ambiguous and hard to deal with in computational systems. To solve this gap, Mally intended to create an exact system of pure ethics to formalize legal and normative reasoning. That was the first attempt at creating a *Deontic Logic* [Lok99], based on the *classical propositional calculus*. Von Wright [vW51] presented a formalism based on *propositional calculus* that was similar to a normal modal logic. Adapting Von Wright's proposal, the *Standard System of Deontic Logic* or *KD* was created as a modal logic with the following axioms:

$$\begin{array}{ll}
 O(p \rightarrow q) \rightarrow (O(p) \rightarrow O(q)) & \text{(KD1 or K-axiom)} \\
 O(p) \rightarrow P(p) & \text{(KD2 or D-axiom)} \\
 P(p) \equiv \neg O(\neg p) & \text{(KD3)} \\
 F(p) \equiv \neg P(p) & \text{(KD4)} \\
 p, p \rightarrow q \vdash q & \text{(KD5 or Modus Ponens)} \\
 p \vdash O(p) & \text{(KD6 or O-necessitation)}
 \end{array}$$

where O , P and F are modal operators for *obligation, permission* and *prohibition*.

Thus Deontic Logic allows for expressing a norm as the obligations, permissions, and prohibitions that an entity has towards another entity. Verbs such as *should* or *ought* can be expressed as modalities: "*it should happen p*", "*it ought to be p*". The semantics of the O , P and F operators define, for a normative system and in terms of possible worlds, which situations are ideal.

However, Deontic Logic is not a logic of norms, but a logic of propositions stating the existence of norms [MP99].

²Some countries, like the United Kingdom, have their laws based on *Common Law*. As opposed to Roman Law, Common Law is composed by an aggregation of past decisions, which are used as a base for future decisions, in a case-based approach. Therefore, there is no constitution nor a layered system of norms.

Norms in Computer Science

The study of norms is particularly interesting for the research in Multi-Agent Systems for its relationship with the study of coordination. However, there is no consensus of what norms are, how they should be modelled, and how agents should reason about them. Virtual organizations and electronic institutions are concepts that have been developed in this field of research.

Some issues still open in this direction are:

- How agents communicate, understand, and fulfill the expectations of the others.
- How the collective behavior emerges from the composition of individual behaviors.
- How to formalize normative systems in a computational representation.
- How to define mechanisms, similar to the ones existing in human societies, to ensure trust in open systems and complex environments.
- How the norms can be incorporated in an organization, in a way that the individuals behave according to them.

Trust, for example, can emerge by incorporating a set of norms in the organization that indicate the type of behavior each of the parties in the transaction should follow, therefore creating an *institution*. We can say then that institutions are established to regulate the interactions among agents that are cooperating.

There are two different points of view on how the social interaction is performed in an institution [VS04]: from the *individual's point of view* or from the *society's point of view*.

Social interaction from the individual's point of view. From this point of view the main concerns are:

- How does it react to signals from other parties?
- Does it have a fixed protocol to handle interactions or does it reason about the signals?
- What is the *best* way to react to a signal?

Within this view, research focuses on the study of norms and their effects in agents' behavior. The former includes work on how to formalize norms and defines logics expressive enough to model an agent's *accepted behavior* through norms, while the latter deals with concepts like how norms affect an agent's behavior, and how an agent can reason about norms.³

Conte and Castelfranchi [CC01], for example, propose an extension of Cohen and Levesque's "*Theory of Rational Action*" [RL87], extending the *BEL*, *GOAL*, *HAPPENS* and *DONE* modal operators with a new operator *OUGHT*, in order to include obligation as another operator inside the formalism. The resulting formalism allows them to express concepts such as *normative belief* (*N-BEL*), *normative belief of pertinence* (*P-N-BEL*) and *normative goal* (*N-GOAL*).

An alternative is the use of Deontic Logic [MW91]. Standard Deontic Logic or *KD* is expressive enough to analyse how obligations follow each other and is useful to find possible paradoxes in the reasoning. However, the *KD* system can hardly be used from a more

³For the purpose of this document, we will focus on the norm formalization rather than in agent's behavior and reasoning about norms. For more information about this specific issue and Norm Autonomous Agents, please check [VS04]

operational approach in order to, e.g., decide which is the next action to be performed, as it has no operational semantics. One interesting extension of KD Deontic Logic is the Dyadic Deontic Logic, proposed by Von Wright, which introduces *conditional obligations* with expressions such as $O(p|q)$ (“*p* is obligatory when condition *q* holds”).

There are also specific logics to address temporal aspects, as the Temporal Deontic Logic. For instance,

$$O(p < q)$$

states that “*p* is obligatory before condition *q* holds”. Another option is combining deontic operators with Dynamic Logic:

$$[p]O(q)$$

means “after *p* is performed it is obligatory *q*”.

Some researchers have enhanced deontic logic by adding the action modal operators E, G, H . Operator E comes from Kanger-Lindahl-Pörn logical theory [KS74, Lin94, Pörn74], and allows to express direct and successful operations: $E_i A$ means that an agent i brings it about that A (i.e., agent i makes A to happen and is directly involved in such achievement). Operators G and H were introduced later by Santos, Jones and Carmo to model indirect actions [SC96][SJC97]. Thus, the modal operator G allows to express indirect but successful operations: $G_i A$ means that an agent i ensures that A (but not necessarily is involved in such achievement). The operator H expresses attempted (and not necessarily successful) operations: $H_i A$ means that an agent i attempts to make it the case that A .

Social interaction from the society’s point of view. Research from this point of view has the following basic concerns:

- Is the interaction *fair*?
- Is the interaction *efficient*?
- Are the objectives of the organization *met*?

Research from this specific point of view focuses on the definition and formalization of Social Systems⁴. There are several approaches to the topic: *game-theoretic*, *sociological*, *organizational*, and *logic*.

Shoham, Tennenholtz and Moses [MT95, ST95] presented a *game-theoretic* computable mathematical formalization of agent societies that includes a *normative level*. They define the concept of *Artificial Social System* as a set of restrictions on agents’ behaviors in a multi-agent environment. An Artificial Social System is a mechanism for coordination that reduces the need for centralized control, allowing agents to coexist in a shared environment and pursue their respective goals in the presence of other agents. The formalization they propose is an automata-based model of multi-agent activity that describes the system as a set of agents that, at any given point in time, are each of them in one state. In order to provide the *Artificial Social System* formalism with a clear semantics, they use the concept of *possible worlds*. Their formalization captures the idea of normative systems as a set of socially acceptable goals and states and while it has been used in some setups, it has some drawbacks, e.g. there is no definition of roles, and the system can see the agents’ goals.

Sociological approaches are based on Balzer’s [Bal90] work, where he presents a quite precise model of what he calls *social institutions* which comprises of four dimensions: the

⁴Simulation of Social Systems is also found in the literature. However, we will not focus on it as the formalisms used are not suitable for our purpose.

macro-level, composed by *groups*, *action types* and *characteristic functions*; the *micro-level*, describing the individuals and the actions; the *intellectual representations* that individuals make from the macro- and micro-level structures, either in their minds or in the language they use for communication; and the *social practices*, which are sets of actions that are performed by some groups and are imitated by other groups. This model was further extended by Balzer and Tuomela in [BT01] in order to include concepts such as *obligations* and *rights*. In this extension a *norm* is seen as a *task right system*. In order to do so, individuals can be assigned to *positions*, where each position defines a set of obligations and rights attached to that position⁵. Although this extension allows the definition of norms, it cannot be used in practice to model open systems, as it does not define how norms emerge or who can create them, or how norms are enforced.

Organizational approaches have been quite successful in applying model social order to software agent societies. For example, V. Dignum *et al.* [DWX02][DMWD02] integrated some previous work in order to present a framework composed by three interrelated models:

- The *organizational model*, which describes the structure of the organization in terms of roles and goal distribution among roles.
- The *social model*, which describes the enactment of roles by the agents, regulated by *social contracts*.
- The *interaction model*, which describes the interactions between the agents that populate the society in terms of *interaction contracts*.

In order to give guidance of the possible interactions that may occur inside the organization, the framework defines *interaction structures*. These structures define patterns of interaction in a way that the social goals can be achieved. The difference between an interaction structure and a protocol is that the latter specifies all the steps of the process, one by one, while the former only fixes some landmarks, giving autonomy to the agents to decide how they will achieve each landmark. A drawback of this framework does not explain how to model the norms that the organizational context may impose to the organization, and how they should be included inside the different levels depending on the normative level of abstraction.

Finally, there are some notable examples of *logic* approaches, as for example the work from Governatori *et al.* [GGRS02b][GGRS02a], which extends the idea of *normative influence* with the concept of *channels of deontic influence* in order to express that a given agent either is directly involved in the fulfillment of a norm, or there is a chain of agents linked by normative influences such that one of the agents will fulfill the norm. In order to model these chains, they propose to add a new action operator *EI*: $EI_i A$ means that agent *i* *attempts to make it the case that A*, by creating a channel of deontic influence terminating with *A*. However, there are several reasons that logic approaches are difficult to implement into agent architectures, being a key one that although these formalisms try to model actions by means of modal action logics, those actions are described in a very abstract way: agents are said to ensure or to bring about an action or a state of affairs, but not how that can be accomplished.

⁵The concept of *position* in this approach is equivalent to the concept of *role* from other approaches

1.4.2 Institutions

Human interactions are usually governed by conventions or rules which emerge from the society itself. In fact, all human societies, even the most primitive ones, have had *social* constraints in order to structure and to regulate the relationship between its members (see 1.4.1). *Institutions* are identified by the set of constraints that govern these relationships. This concept of *human institution* can be used and applied to service-based interactions, in what we call *Electronic Institutions*.

Human Institutions

North [NNS90] studied the effect of sets of constraints, that he refers to as *institutions*, on the behavior of human organizations. As seen in Section 1.4.1, North claims that institutional constraints ease human interaction, shaping choices and making outcomes foreseeable. The creation of these constraints allows for a growth on the complexity of the organizations while keeping reduced interaction costs, and equally important, allows the participants of the institution to act, and expect others to act, according to a list of rights, duties, and protocols of interaction.

Therefore, the creation of institutions provides trust among parties even when they don't have much information about each other.⁶ In environments with incomplete information, cooperative interactions can perform ineffectively unless there are institutions which provide sufficient information for all the individuals to create trust and to control deviations.

Institutions can be classified according to how they are created and maintained, or on the formality of its rules. On the former case, institutions can be created from scratch and remain *static* or be continuously *evolving*. On the latter, institutions can be *informal*, that is, defined by informal constraints such as social conventions and codes of behavior, or *formal*, defined by *formal rules*. Formal rules can be *political and judicial rules, economic laws, or contracts*.

In *formal institutions* the purpose of formal rules is to promote certain kinds of exchange while raising the cost of undesired kinds of exchange. Elnor Ostrom [Ost86] classifies formal rules in 6 types:

- *position rules*: to define a set of positions (roles) and the number of participants allowed for each position,
- *boundary rules*: to state how participants are chosen to hold or leave a position,
- *scope rules*: to specify the set of outcomes and the external value (costs, inducements) related to them,
- *authority rules*: to specify the set of actions assigned to a position at a certain node,
- *aggregation rules*: to specify decision functions for each node to map action into intermediate or final outcomes,
- *information rules*: that authorize channels of communication among participants in positions and specify the language and form in which the language will take place (the protocol).

As norms are, in fact, the elements that characterize institutions, they do not only serve as norms to be followed, but also serve as indication for people to recognize an organization as being an instance of a particular kind of institution, and then use this knowledge to predict other norms that could be applicable.

⁶No institutions are necessary in an environment where parties have complete information about others.

Electronic Institutions

We have seen in Section 1.4.2 that the lack of information about the interactions of the members of an organization can penalize the efficiency of these interactions. The same statements hold for closed multi-agent systems, where trust is implicit, and open multi-agent systems, where trust is something that has to be built by some kind of mechanism, such as the *Electronic Institutions*.

An *Electronic Institution (e-institution)* [VS04] is the model of a human institution through the specification of its norms in some suitable formalism. The essence of an institution, through its norms and protocols *can* be captured in a precise machine processable form and this key idea forms the core of the topic of institutional modelling.

Institutions, seen as norm providers and enforcers, intend to solve the following issues in the context of Multi-Agent Systems:

- Reduce uncertainty about other agents' behavior inside the institution.
- Reduce misunderstanding with a common set of norms governing interactions.
- Allow agents to foresee the outcome of a certain interaction among participants.
- Simplify the decision-making process inside each agent, by reducing the number of possible actions.

In summary, the use of norms makes agents more successful in the achievement of their goals.

State of the art on normative frameworks

As seen in Sections 1.4.1 and 1.4.1, there is quite some research on theoretical approaches and methodologies to model and design agent-based social systems. In this Section we will summarize research done on methodologies and frameworks for Electronic Institutions.

Some approaches have been designed specifically for the *e-commerce* and *e-business* fields, such as SMACE [CO00] or the Contractual Agent Societies (CAS) [Del00]. In both cases the normative perspective is modelled by the use of *contracts*, which enforce agents to comply with the agreements they have done. However, these contracts only model agreements between parties, and do not cover those permissions, prohibitions or obligations that an *e-organization* may need to define in order to ensure an appropriate behavior of the society of agents.

The SMART agent architecture presented by d'Inverno and Luck [dL04] is based on an agent specification framework developed on top of the Z specification language [Spi89]. This framework defines concepts such as *objects*, *agents* and *autonomous agents*⁷. This framework was extended by López y López, Luck and d'Inverno [yLLd01][yLL02] by introducing representations of norms, which are not modelled as static constraints but as objects that can have several states: *issued*, *active*, *modified*, *fulfilled* or *violated*. These norms are related not only with the agents that should fulfill or enforce them, but also with agents such as the one that issued the norm, the one that modified it or the ones that may be affected by a violation of the norm. However, there is no reported implementation of the architecture on a real use case.

ISLANDER [Nor97, RA03, EPS02] is a proposal that treats electronic institutions as a type of *dialogical system* where all the interactions inside the institution are a composition

⁷Agents are defined as objects with goals, while *autonomous agents* are defined as objects with motivations.

of multiple dialogic activities, that is, message exchanges. These interactions, called *illocutions* [Nor97], are structured through agent group meetings called *scenes* that follow well-defined protocols. This division of all possible interactions in scenes allows for a modular design of the system. Another important element of ISLANDER is the notion of *role*: each agent can be associated to one or more roles, and these roles define the scenes the agent can enter and the protocols it should follow. ISLANDER has been mainly used in *e-commerce* scenarios, and was used to model and implement an electronic Auction house, the *Fishmarket*. The resulting agent-mediated auction house was used, to compare different strategies of the trading agents [CRA00][GGGRA99][MS99]. Still, ISLANDER presents some drawbacks in its application to complex domains, due to its *e-institution* modelling in a low level of abstraction, by means of constraints on behavior modelled through very precise step-by-step communication protocols, thus reducing the autonomy of the agents.

Finally, HARMONIA[VSD03] is a framework that defines a multi-level structure, from the most abstract level of the normative system to the final implementation of an electronic organization. In this framework, norms are defined to the final protocols and procedures that implement them. The main objective of HARMONIA is to fill the existing gap in previous approaches, which work either at the level of norm formalization or at the procedural level. Some important parts of this model are formalized using multi-modal logics. HARMONIA is the formalism chosen for the purpose of this PhD proposal, so a more detailed description of it can be found in Section 4.1.

1.5 OBJECTIVES

The main idea of this PhD proposal is to incorporate an existing framework for Electronic Institutions in the SOA governance methodologies. An immediate and obvious challenge that this objective arises is the need to adapt the chosen Electronic Institutional framework in order to be used in generic Service-Oriented Architectures.

Electronic Institutional frameworks are currently built on top of Multi-Agent Systems. Agents can be used as a form of distributed computation that fits, as a subset, in the scope of Service-Oriented Computing [SH05]. On the other hand, SOA Governance does not focus on MAS either.

Therefore, the adaptation that we are proposing is, in fact, an abstraction. With our architecture proposal we aim at bridging this gap by combining Web services and agents inside a normative framework derived from HARMONIA, and deployed in heterogeneous (MAS, Web services and/or Grid) platforms.

In most cases, the components of an *e-institution* based on MAS can be integrated into a SOA without much effort, in terms of connectivity. But there are at least two important exceptions that make this generalization required. The first one is the capture of the events of the system, essential to detect violations and enforce the norms. The other one is the norm enforcement mechanism, as well as the language associated to this mechanism. Our previous research is directly related to these two abstractions.

Concerning the observation of SOA interactions, some contributions have been done, specifically in the EU-Provenance project. The research done in this field include the integration of a SOA monitoring formalism and architecture in a practical use case. In the case of the norm enforcement, research has been done in creating a formalism for electronic contracts in SOA, along with a middleware for the creation of electronic contractual institutions using services with reasoning capabilities.

With the research done in mind, we introduce here a proposal for an architecture for norm enforcement in SOA, which will be based on Provenance as monitoring mechanism and using an electronic language for norms, on top of service agents. This architecture will be implemented and tested in SOA governance processes. This will allow for testing enforcement and governance in SOA, and to 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.

Therefore, the main objectives of this proposal are:

- To design and implement an open source norm enforcement framework using the latest technology based on SOA available.
- To apply and generalize, on the design of this framework, the state of the art in Electronic Institutions.
- To integrate this framework in the SOA governance methodology lifecycle.
- To create a mapping between the operational representation of norms and both orchestration and choreography languages.
- To benchmark the resulting framework against other SOA governance options available in the industry.
- To find and integrate the ideas and concepts shared by both Governance and Institutional theoretical frameworks but developed independently.
- To adapt the existing theories about Electronic Institutions to the wider scope of SOA.
- To contribute the advances done in regard to the SOA generalization to the theory of Electronic Institutions.

1.6 USE CASE: THE OTM APPLICATION

Patient treatment through the transplantation of organs or tissues is one of the most complex medical processes currently carried out, as it is a distributed problem involving several locations (donating hospital, potential recipient hospitals, test laboratories and organ transplant authorities, see Figure 1.1), a wide range of associated processes, rules and decision making. It is recognized worldwide that IT solutions which increase the speed and accuracy of decision making could have a very significant positive impact on patient care outcomes.

1.6.1 Distributed Transplant Management: the OTM 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 [VSCP⁺03], the aim of which was to speed up the allocation process of solid organs to improve graft survival rates. Its policy implements the Spanish guidelines for organ and tissue procurement and Spanish regulations for allocation, as Spain is world leader in the area, followed as a model by other countries. OTM uses standard web service technology and has been adapted to be provenance-aware (see Chapter 2), by interacting with the provenance stores in order to keep track of the distributed execution of the allocation process for audit purposes.

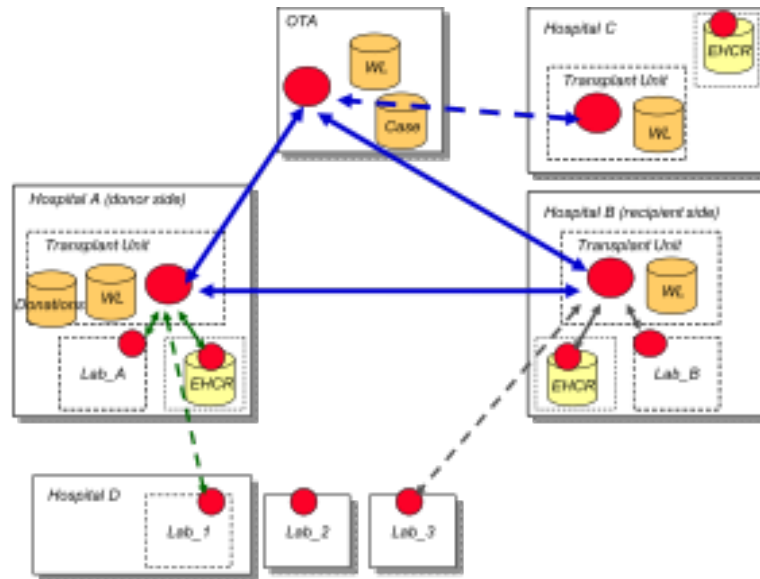


Figure 1.1: **The OTMA system.** Each medical unit is represented by an agent (circle in figure) which manages interactions with other units and with the EHCR subsystem.

Figure 1.1 summarizes the different administrative domains (solid boxes) and units (dashed boxes) that are modelled in the OTM application. Each of these interact with each other through Web Service interfaces (circles) that send or receive messages. The Organ Transplant Authority (OTA) is an administrative domain with no internal units. In a transplantation management scenario, one or more hospital units may be involved: the hospital transplant unit, one or several units that provide laboratory tests and the unit that is responsible for the patient records. The diagram also shows some of the data stores that are involved: apart of the patient records, these include stores for the transplant units and the 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 its own records of each donation, stored case by case.

1.7 STRUCTURE OF THIS DOCUMENT

The next two chapters describe the work already done that will be useful for the achievements of the goals defined in Section 1.5.

First, in Chapter 2, we present our contributions done in the observation of SOA interactions, focusing on Provenance. Then, in Chapter 3, some contributions about SLA into Contractual Institutions are summarized, which cover the part of the enforcement in SOA.

In Chapter 4, we introduce a proposal of an architecture for norm enforcement in SOA based on Provenance, which is a first step on the way of generalizing SOA Electronic Institutions.

Finally, Chapter 5 describes the estimated working plan for the PhD proposed.

Behaviour Monitoring in SOA: Provenance

As seen in Section 1.5, monitoring is one of the important points of abstraction for *e*-institutions to be deployed and integrated in SOA. In this chapter we explain the contributions made in monitoring in SOA, concretely in the handling of Provenance¹.

Monitoring, in its *traditional* meaning in Software Engineering, refers to the provision of information, usually by *sensors*, about the system's environment, in order to take actions depending on the result of some processing done to this information [Som06]. In other words, monitoring is based on the logging, keeping, and interpretation of messages sent by components of the software system.

There are many monitoring mechanisms available in most of the service deployment platforms, which cover this need. However, with the growth in the complexity of the platforms based on SOA, and especially with the Grid, it has been shown that this definition of monitoring is not enough to deal with common problems in distributed scenarios [MKR⁺07, GLM04].

That is also the case of *e*-institutions, where the simple logging of messages, not explicitly linked to each other, is not enough. A norm enforcement mechanism needs more information about the distributed scenario, e.g. the actual relationship between interactions, which are sent at different points in time and thus are apparently not related to each other, but are in fact parts of a complex transaction.

Provenance mechanisms try to fill this gap. Rather than focusing on capturing interactions and message exchange in a free format, capturing provenance allows for the capture of internal states of the actors as well as the relationship between these interactions, and the relationship between states with interactions or other states. Additionally, these recordings contain semantic information that allows for unambiguous interpretation. Therefore,

¹Part of the research described in this chapter has been published in [KVVS⁺06, ANVSK⁺06, KVAN⁺06, VSANK⁺07]

with provenance handling we have mechanisms that provide interpretable monitoring of loosely-coupled distributed complex processes.

2.1 PROVENANCE

The concept of *provenance* [VSANK⁺07] is already well known in fine art where it refers to the trusted, documented history of some work of art. Given that documented history, the object attains an authority that allows scholars to understand and appreciate its importance and context relative to other works of art. Objects that do not have a trusted, proven history may be treated with some scepticism by those that study and view them. This concept of provenance may also be applied to data and information generated within a computer system, especially when the information is subject to regulatory control over an extended period of time.

2.1.1 Related Work to Provenance

In the first investigations which started to record the origin and history of a piece of data, the concept was called *lineage*. In the SDTS standard [GJM⁺06], lineage was a kind of audit trail that traces each step in sourcing, moving, and processing data, mainly related to a single data item, a logical data record, a subset of a database, or to an entire database [BKT01, WS97]. There was also relationship to versioning [MACM01] and data warehouses [CWW00].

The provenance concept was later further explored within the GriPhyN project [VWF02]. These techniques were used in [FVWZ03] in two respects:

- Data was not necessarily stored in databases and the operations used to derive data items might have been arbitrary computations.
- Issues relating to the automated generation and scheduling of the computations required to instantiate data products were also addressed.

The PROVENANCE project [MI06] built on these concepts to conceive and implement industrial strength open provenance architecture for grid systems, including tools for managing and querying provenance stores along with high-level reasoning capabilities over provenance traces. The price to pay for this, as we will see in Section 2.2, is that applications should be adapted in order to provide high-quality assertions that not only record their inputs and outputs but also the causal and/or functional relation between them.

The alternative would be the use of automatic data harvesting techniques such as RDF tuples harvesting [Fut06], where RDF tuples include attribution (*who*) and time (*when*) information which is then processed by an external inference engine in order to construct RDF graphs by some kind of extended temporal reasoning.

Another alternatives reduce to a minimum the adaptation step needed to make an application provenance-aware by adding a middleware layer or an execution platform capable to automatically create provenance assertions from the captured events and actions [BD06, MRHBS06]. The problem is that, in automatic provenance collectors, it is very hard to infer causal relationships by only comparing sources and times [BGH⁺06], sometimes with the extra help of some derivation rules [FVWZ02] or rigid workflow definitions, placing a costly constraint over the adaptation.

2.1.2 Provenance in Service Oriented Architectures

The contribution done in this field has been to adapt the use case described previously in Section 1.6, where there was a need to collect the electronic trace of the medical history of patients, in the scope of the EU-Provenance project [MI06] which studied the provenance of electronic data in SOA. In the following we are going to describe provenance based on the Provenance Architecture document [GJM⁺06].

The aim of the Provenance project was to conceive a computer-based representation of provenance that allows users to perform useful analysis and reasoning. The provenance of a piece of data can be represented in a computer system by some suitable documentation of the process that produced the data. This documentation can be complete or partial, 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 project assumed that provenance is investigated in open, large-scale systems typically designed using a service-oriented approach. Services are regarded as components that take inputs and produce outputs. Such services are brought together to solve a given problem typically via a workflow that specifies their composition. In this abstract view, interactions with services (seen as *actors*) take place using messages that are constructed in accordance with service interface specifications.

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. The technology-independent approach of the Provenance project to service-oriented architectures (SOAs) has formal foundations in the *pi*-calculus[Mil99] and asynchronous distributed systems[Lyn96]. According to this view, messages are the only mechanism used to transfer information between actors. The *pi*-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 us to formally define mappings with agent-mediated services and to use the Provenance project results for Multi-Agent Systems.

The Provenance project developed an architecture, tools and a reference implementation to support this provenance lifecycle.

2.1.3 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 out-put 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.

The *provenance lifecycle* is composed of four different phases. First, actors create p-assertions that are aimed at representing 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 an specific interface (subscription management, content relocation, etc).

By transforming a distributed system into a provenance-aware system, the resulting system gets the capability to produce at execution-time an explicit representation of the distributed processes that are taking 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.

2.2 APPLYING PROVENANCE IN A DISTRIBUTED SOA-BASED SYSTEM

The use of ICT solutions applied to Healthcare in distributed scenarios should not only provide improvements in the distributed processes and services they are targeted to assist but also provide ways to trace all the meaningful events and decisions taken in such distributed scenario.

Cooperation among people using electronic information and techniques is more and more common practice in every field including healthcare applications as well. In the case of distributed medical applications the data (containing the healthcare history of a single patient), the workflow (of the corresponding processes carried out to that patient) and the logs (recording meaningful events) are distributed among several heterogeneous and autonomous information systems. These information systems are under the authorities of different healthcare actors like general practitioners, hospitals, hospital departments, etc. which form disconnected *islands of information*. In order to provide better healthcare services, the treatment of the patient typically requires viewing these pieces of workflow and data as a whole.

Also, having an integrated view of the workflow execution and the logs may become important in order to analyse the performance of distributed healthcare services, and to be able to carry out audits of the system to assess if needed, that for a given patient the proper decisions were made and the proper procedures were followed. For all that there is a need to be able to trace back the origins of these decisions and processes, the information that was available at each step, and where all these come from.

In our agent-mediated healthcare system OTM application (see Section 1.6), the different human actors coordinate with each other in the execution of a medical process through a distributed system based on agentified web services. By recording all the medical processes related to a given patient one can reconstruct the treatment history of the patient. Therefore, making an agent-mediated healthcare system provenance-aware provides a way to have a unified view of a patient's medical record with its provenance, i.e. to con-

nect each part of the medical record with the processes in the real world that originated it and/or the individuals, teams or units responsible for each piece of data.

Therefore, provenance is an innovative way to trace such events and decisions in Distributed Health Care Systems, by providing ways to recover the origin of the collected data from the patients and/or the medical processes. In this Section we present our work done to apply *provenance* in the domain of distributed organ transplant management.

2.2.1 Provenance Handling in the OTM Application

Making the OTMA system provenance aware presented three challenging issues: a) the provenance of most of the data is *not* the execution of computational services, but decisions and actions carried out by *real* people in the *real* world (this is discussed in Section 2.2.2); b) past treatments of a given patient in other institutions may be relevant to the current decisions in the current institution, so information of the processes undertaken in those previous treatments should be connected to the provenance information of a current process (this is discussed in Section 2.2.3); c) the agent with provenance information knows much more about the patient than any other agent in the system, so there are privacy risks to be mitigated (this is discussed in Section 2.2.4).

2.2.2 Adapting the OTMA system for Provenance

As seen in Figure 2.1, in the OTM application, each organizational unit (the transplant unit, the ER unit, the laboratories) is represented by an agentified service. Staff members of each unit can connect to the unit services by means of GUI interfaces. The provenance of a data item is represented by a set of *p-assertions*, documenting steps of the process, and they are stored and managed in *provenance stores*. The distributed execution of the OTM services is modeled as the interaction between the actors representing the services, and recorded as *interaction p-assertions* and *relationship p-assertions*. As in the OTM scenario a decision depends on a human making the decision, additional *actor state p-assertions* are recorded, containing further information on why the particular decision was made and, if available, the identities(s) of the team members involved in the decision.

To illustrate how provenance is handled in the OTMA system, let us see how the provenance of a medical decision is recorded.

Figure 2.1 shows the OTMA agents for this small scenario and their interactions. The Transplant Unit User Interface Agent passes requests (TU.1, TU.2) to the OTM Donor Data Collector Agent, which gets the electronic record from the EHCR system (OTM.1, OTM.2). Sometimes all or parts of the record are not in the same institution but located in another institution (HC.1, HC.2). The Donor Data Collector Agent also sends the request for a serology test to the laboratory and gets back the result (OTM.4), along with a detailed report of the test. Reports are also passed in the case of the Brain Death notification (TU.3) and the final decision report (TU.5).

Figure 2.2 graphically represents the subset of the p-assertions produced by the *provenance-aware* OTMA which are related to the mini-scenario described above. The part of the process that happens within the electronic system is represented by interaction p-assertions (regular boxes) for all interactions (TU.x, OTM.x, HC.x), and relationship p-assertions (*response_to*, *caused_by*, *based_on*) capturing dependencies between data. Even though what happens in the system parallels what happens in the real world, as we already said this is not enough to fully determine the provenance of a given decision. To solve this, we connect the electronic process to the real world by adding actor state p-

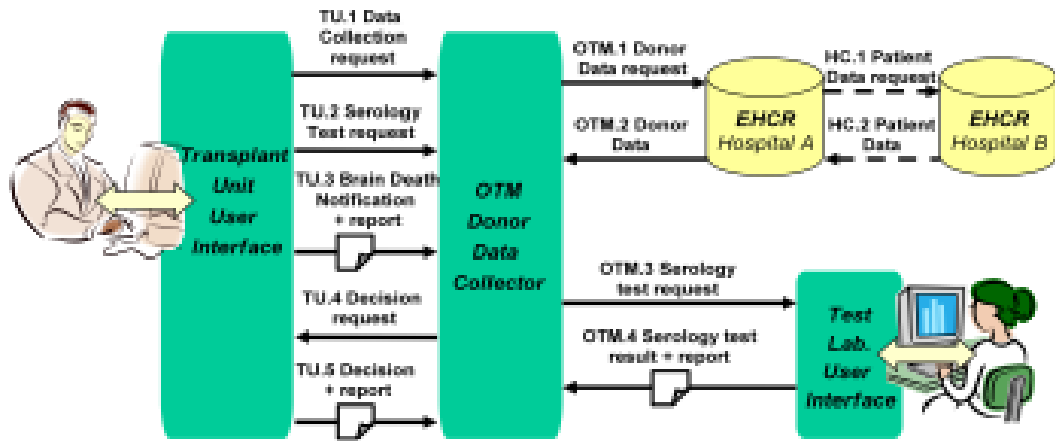


Figure 2.1: **Example scenario:** Interactions of the OTMA agents involved in a donation decision.

assertions stating who logged the information in the system (*is_logged_in*) and when (not shown in picture), which are the reports that justify a given state in the system (*justified_by*), who are the authors of these reports (*authored_by*) and when the action reported was performed or the decision taken (not shown).

Provenance Questions

In both the OTM and the EHCR systems, the provenance architecture should be able to answer the following kind of questions, related to a given patient (donor or recipient) or to the fate of a given organ:

- where did medical information used on each step of the process came from
- which medical actor was the source of information
- what kind of medical record was available to actors on each step of the process
- when a given medical process was carried out, and who was responsible for it
- when a decision was taken, and what was the basis of the decision
- which medical actors were asked to provide medical data for a decision
- which medical actor refused to provide medical data for a decision

All these kind of questions can be answered by querying the provenance store. A query will give as a result (a subset of) the provenance representation graph of the process related to the query. If we use as an example the graph in Figure 2.2, by following the edges from the “Donation Decision” p-assertion we can trace the provenance of the donation decision, how it was based on some data and test requests, how a brain death notification is also involved, who requested the information, where it came from (in some cases it might come from the EHCR of another hospital), and who authored the justifying reports in the main steps of the process.

In those cases (as in Figure 2.1) where the decision might be based on medical data coming from tests and medical treatments carried out in other institutions, another issue to solve is the following: how to find, retrieve and incorporate the provenance of the data

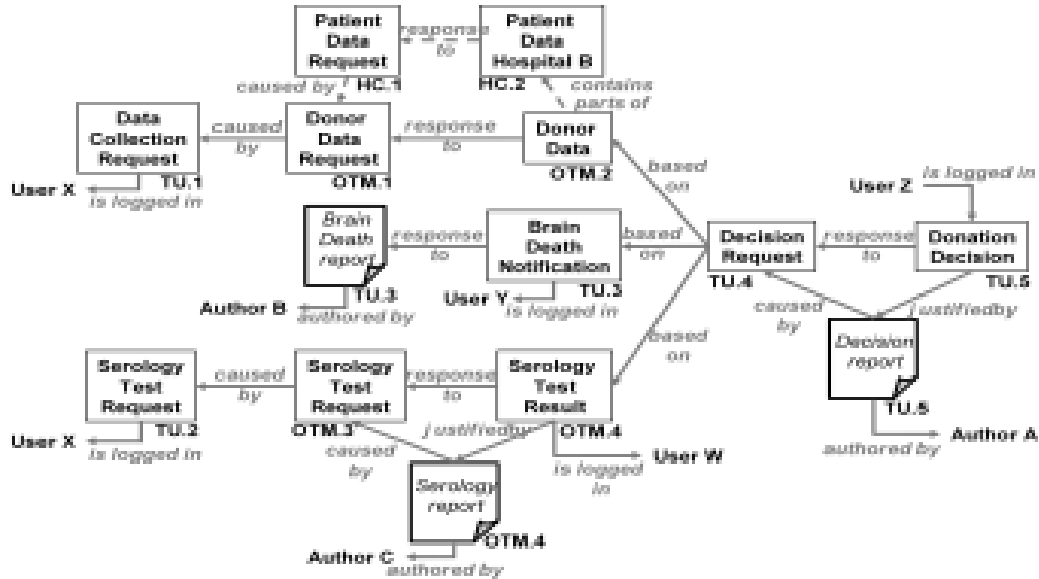


Figure 2.2: Example scenario: Directed acyclic graph showing the provenance of the donation decision.

coming from the other institution? If these institutions have also provenance-aware systems and the provenance stores of the different institutions are connected, to solve the aforementioned problem is to solve the issue of matching the different p-assertions related to the same patient. If this match is done, then actors can make p-assertions that link together the separate sets of p-assertions to create a larger provenance document providing an integrated view of the healthcare history of the patient. The result (not shown on Figure 2.2) would be that the p-assertions related to *Patient Data Hospital B* would be linked to the set of p-assertions already part of the provenance of the Donation Decision.

Collectively the p-assertions can be seen as describing a distributed process, spanning space as well as time. Every relationship described is causal, i.e. between the cause of something happening and the effect of it happening, and is therefore also temporal, i.e. causes always come before effects. Furthermore, extra information can be added to provide further detail. For example, an actor may record, as an actor state p-assertion, the time shown on their local clock. Together, the structured documentation of processes allow a rich set of questions to be asked about what occurred, why, when and by whom and, in the OTMA system, such a process may be a patient’s healthcare history.

2.2.3 Process Documentation in Provenance Aware HC-MAS

As seen in the previous section, in order to create an integrated view of a patient’s healthcare history, the *process documentation* created by each healthcare institution must contain interaction p-assertions and relationship p-assertions which link together the p-assertions of agents in the process. The way in which p-assertions provide this linking in usual service-oriented applications is by use of a common identifier, called an *interaction key*,

for both parties, sender and receiver, in an interaction. From the fact that two agents have recorded documentation using the same interaction key, we can determine that their actions were part of the same process, and therefore both are part of the provenance of the process' output. However, to record p-assertions with the same interaction key, two agents must exchange that key, which means they must electronically interact.

In this section we show that there are processes without direct electronic interaction. Because processes executed by healthcare agents often belong to this category, we describe how independent autonomous agents can jointly create process documentations of these processes.

Healthcare Processes as Weakly Connected Processes

In a typical business or e-science application the agents participating in the process are in contact and connected by interactions between them. In this case there is exchange of documented messages, and we say that there is *direct interaction* between the agents.

In medical processes, however, the physicians treating the same patient may not be in direct contact. This is typical, but not necessarily specific to medical processes. The patient may be treated by one physician, be healthy for a while, and then go to another physician with another disease, in some cases as a consequence of the previous disease. In this case, the second physician is not in contact with the first, they do not know each other's identity and because the identities cannot be revealed for privacy reasons, the p-assertions belonging to the same patient cannot be linked together automatically. In this case we say that there is *latent interaction* between the physician agents. Note that the patient usually cannot determine the link between the current treatment and the previous one. Even if the patient remembers something informally, the formal link cannot be determined.

We can now define two types of processes: strongly connected and weakly connected processes. We view the processes as graphs where the nodes are the activities executed by agents alone and arcs are the interactions, either latent or direct. In *strongly connected processes* the whole process graph contains only direct interactions, whereas in *weakly connected processes* the process graph can be cut into two sub-graphs that are connected to each other only by one or more latent interactions. The full healthcare history of a patient is usually created by a weakly connected process containing strongly connected sub-processes.

Process Documentation of Strongly Connected Processes

Figure 2.3, similarly to Figure 2.2, shows the model of strongly connected processes and their process documentation. Here, physicians are represented by agents 1 and 2. They are the actors of treatment processes 1 (treatmentp_1) and 2 (treatmentp_2). At some point, agent 1 sends the patient to agent 2 in a documented way. P-assertions about this interaction are recorded by agents 1 and 2. In case of medical or other secure applications, a global identifier for a patient in the local system is not used, because it could be used to determine the identity of the patient. Because the agents interact directly and electronically, they agree on an interaction key and both include it in their p-assertions. This way the process documentations of the treatment processes are not disjoint, therefore if some agent queries the process documentation using patient_local_name_1, then the provenance system is able to return all process documentation comprising the provenance of the patient.

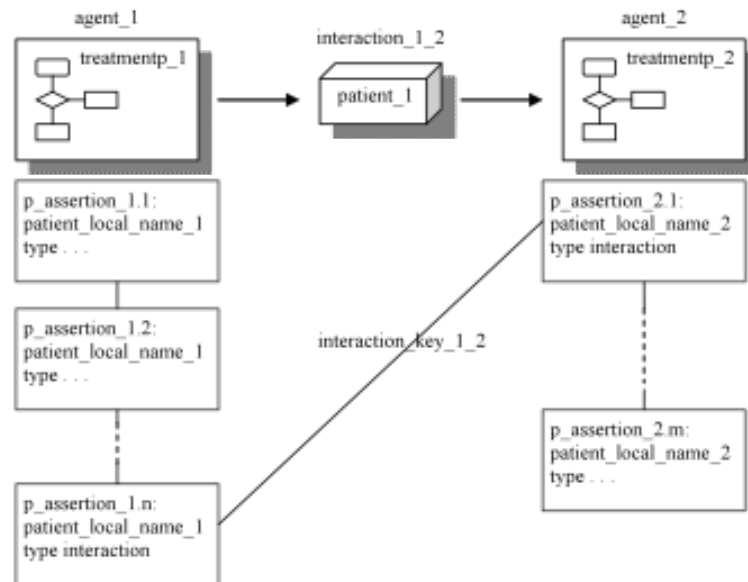


Figure 2.3: **Process documentation in strongly connected processes.** P-assertions of strongly connected processes are linked together by the p-assertions related to the interaction connecting the two processes.

Process Documentation of Weakly Connected Healthcare Processes

Process documentation creation is a bit more complicated in the case of weakly connected healthcare processes, when there is no direct interaction between the agents. The actors, processes and process documentation in a weakly connected healthcare process are similar to the one shown in Figure 2.3, but there is no link across the sets of p-assertions of the processes executed by the different agents. If we want to retrieve the complete provenance of the patient, then we are interested in both sets. Moreover if agent 2 finds out somehow that treatment process 2 is some way a consequence of treatment process 1, it still cannot find the relevant p-assertions made by agent 1, because there is no common identifier for the processes or the patient.

If there was interaction between the agents, then the common identifier could be the common interaction key which they share. However an interaction key is used to identify the flow of information between two steps/actors in a process, and is an internal identifier, in that it has no meaning outside of the process documentation and while, theoretically, it corresponds to one patient wherever the information exchanged in an interaction concerns just one patient, there is no mapping held in the system between an interaction key and any other patient identifier.

Although the patient could present its global identifier, such as its social security number, to the physicians, but this global identifier cannot be used in process documentation for privacy reasons as discussed in Section 2.2.4.

Method to Link Process Documentations of Weakly Connected Processes

We have described the problem of process documentation creation resulting from the lack of direct interaction between the agents. The solution to the problem is to use an intermediate institution in a higher hierarchical level, which is in contact with both agents and knows about the patient as well. This is usual in medical domains, as they are regulated by national and international bodies and there are services which give a global identifier to the patient, such as the national security number. But the global identifier should not be used in documentation of privacy-aware processes, because regulations ordain the separation of data, which means that medical information and personal identification cannot be stored together and anonymized identifiers must be used instead. Therefore, usually we should add an anonymization service in HC-MAS to convert real patient identifiers to anonymized patient identifiers.

Figure 2.4 shows how this method works. In the first step of this method, we locate in the application an already existing service which is used to anonymize patient identifiers. If there is no such service, then we introduce it into the application. The service is called `anon_service` in the figure.

The second important element of the method is that the anonymization service documents its own processing. Whenever a new patient identity becomes known to the anonymization service, then the anonymization service puts an actor state p-assertion into the provenance store about the patient identity. Note that the provenance store does not contain the global patient identifier, only the anonymized identifier.

The third important element of the method is that the agents adopt the norm of notifying the anonymization service when a new activity with a patient is started. In Figure 2.4, when agent 1 starts an activity on the patient, it makes an actor state p-assertion about the start of the activity and notifies the anonymization service that the activity started. The interaction is recorded in the provenance store with interaction p-assertions on both sides. The anonymization service asserts a relationship p-assertion between the p-assertion related to the anonymized patient identity and the p-assertion related to interaction between agent 1 and the anonymization service. When agent 2 starts an activity on the patient, it behaves similarly, therefore there will be an indirect link between the two agent's processes, and the complete provenance of the patient record can be determined.

Although conceptually the anonymization service becomes somehow a central interaction node in the system, scalability can be maintained. Agents communicate limited amount of data with the anonymization service only when they start a new case. The anonymization service creates a new p-assertion for the case, and agents link further p-assertions to the start case p-assertion without communicating with the anonymization service. The functionality of the anonymization service can be distributed in real implemented systems among cooperating services allocated to countries, regions, insurance companies, etc.

In addition to the ability to return the whole process documentation, the method described above has two other advantages: the agents can improve the quality of the process documentation and of their own activities.

The quality of the process documentation can be improved if the agents discover some relationship from the real processes (e.g. the current illness of the patient is a consequence of a problem in the previous treatment not discovered before). In these cases, they can augment the existing links documented by the anonymization service with direct causal

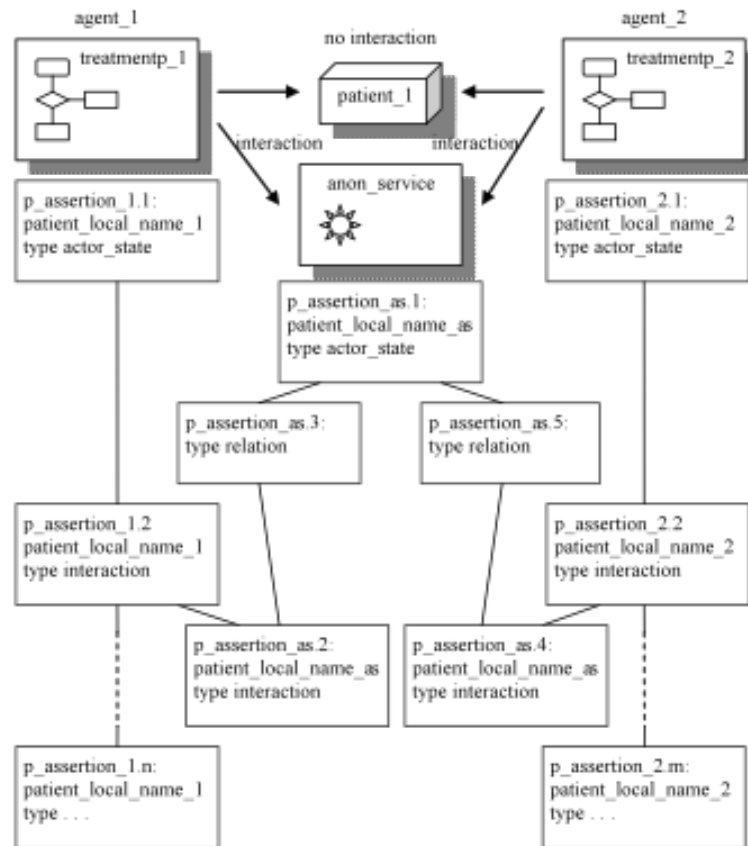


Figure 2.4: **Linking of process documentations in weakly connected healthcare processes.** P-assertions of weakly connected healthcare processes are linked together by the p-assertions of a higher level service.

relationships. This is now possible, because, by following the anonymization service based links, the p-assertions relevant to a single patient can be located, identified and linked. The agents can improve the quality of their own activities using the linked process documentation, because when agent 2 executes its treatment process, it can already retrieve the p-assertions of agent 1. If the physician knows the details of the previous treatment of the patient, then he/she might use that information in the current treatment.

2.2.4 Protecting Privacy in the Provenance-aware Application

In healthcare applications, enforceable *privacy* rules are extremely important. Protection of individuals' health-related data has been a continued concern of the medical body from the very beginning of the medical practice, as reflected in the famous Hippocratic oath. There exist considerable efforts to put into practice a body of policies which ensure the protection of medical data in a scenario of massive use of computers in the health sector.

Regulations² define guidelines about the adequate organizational and technical measures that must be taken in medical information systems. One of these guidelines is related to the separation of data: as a general rule, the design of data structures, procedures and access control policies must be such that they allow the separation of a) identifiers and data related to a person's identity, b) administrative data, c) medical data, and d) genetic data. Such separation must ensure that no unauthorized person can connect the identity of the patient with his medical or genetic data.

A typical solution for the separation of identity information and medical data is the *anonymized identifier*. The anonymized identifier is generated from real patient identifier, and medical data is stored together with this anonymized identifier. If we know the real patient identifier, then we can find the corresponding medical data, but from the medical data we cannot find out the identity of the patient.

When we make agent systems provenance-aware, we introduce an additional data store into the system: the provenance store. There is a conflict between provenance and privacy. While for provenance we need as much information as possible about the whole process (*who* did *what* and *when*), for privacy we need to restrict as much as possible the information available, in order to avoid identification of patients and practitioners by unauthorized users.

In the provenance aware OTMA system two techniques are used to protect privacy: a) we do not store sensitive medical data in the provenance store, and b) we use anonymized patient identifiers in provenance stores.

In order to protect medical data, agents do not store sensitive medical data in the provenance store, but only references to such data. This way the provenance store contains only the linkage and the skeleton of the provenance of the medical data, and the healthcare data can be laid on the skeleton by retrieving it from the healthcare information system when needed. With this approach we keep the same degree of privacy of medical data as in the original distributed system.

One might think that if we do not store medical information about patients in the provenance store, then there is no need to anonymize the patients and we can use real patient identifiers, because no medical information can be inferred about the patient. However this is not the case. Even the fact that the patient was treated can be sensitive information, and the reference to the place where the medical data of the treatment was carried out may contain sensitive information, because the type of institution can reveal the type of medical intervention. Therefore the patient identity has to be anonymized at least.

The anonymization procedure should be irreversible: nobody should be able to tell the real identity of the patient by knowing the anonymized identifier. The irreversibility of the anonymization is guaranteed by the way data storage is organized: the anonymization service does not store the mapping from the real patient identifier to the anonymized patient identifier and computes the anonymized identifier each time it is needed using its own non-trivial algorithm. As a result, the real identifier and the anonymized identifier are not stored together anywhere in the system and the mapping from one identifier to the other cannot be found out without the algorithm of the anonymization service.

²"Directive 95/46/CE of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and of the free movement of such data," Official Journal of the European Communities, L 281/31 - L 281/39., October 1995.

2.3 CONCLUSIONS

In this Chapter we have seen how the integration of provenance in a complex distributed scenario helps solving some of the issues found in standard monitoring mechanisms, especially those based solely on message passing.

In Section 2.1.1 we summarized several alternatives which also go on this line of research. The contributions made by the PROVENANCE project have been introduced in Section 2.1.2. The architecture, as seen in Section 2.1.3, provide three kinds of representation for the documentation (p-assertions) that allow for the interpretation and reconstruction of complex processes.

This architecture has been successfully integrated into our distributed system use case, as described in Section 2.2.2. We have seen how the different types of p-assertions allowed us for describing all the relevant events of the organ transplantation process, storing as a documentation parts of processes that could be, depending on the part of the medical protocol, strongly or weakly connected. This documentation can then be queried and reconstructed, as seen in Section 2.2.3. Finally, in Section 2.2.4 we have shown how the privacy can be ensured in a provenance-aware distributed system.

The features presented in this Chapter are of considerable interest for our purposes. Norm enforcement requires a mechanism that allows for the retrieval by query, and storing, of any relevant semantically compliant element of a complex process in distributed scenarios, to be able to reason about the actual state of the institution and execute a reasoning process around it. As Provenance provides an architecture that addresses these requirements in heterogeneous distributed scenarios, this will be our monitoring mechanism of choice for our proposal in Section 4.3.

Behaviour Enforcement in SOA: From SLA to Electronic Contractual Institutions

In this Chapter we are focusing on *behavior enforcement in SOA*. The literature on this topic is ample and presented in Section 3.1. However, as we will see in Section 3.2, they do not fully cover the requirements for an *e-institution* for SOA, and we will describe the concept of *contractual institutions*. This will lead to an explanation about a language that fits this kind of institutions in Section 3.3, and a middleware that can create them in Section 3.4, both contributions made in this field of research¹.

3.1 SLA IN SOA

The concrete approaches used for enforcement in SOA are based on Service-Level Agreements (SLA) [ANOB⁺07]. SLA is a formal negotiated agreement between a service provider and his customer. When a customer orders a service from a provider, an SLA is negotiated and then a contract is drawn up. The service provider must perform a SLA monitoring in order to verify whether the Quality of Service (QoS) parameters specified in the SLA contract are respected. The SLA monitoring involves monitoring the performance status of the offered service and provides relevant information to the service level management system. Then, the system management assesses the provider's commitments and applies penalties if those commitments were not met.

In order to define such provider's commitments, a lot of specification works has been recently carried out defining several XML based languages enabled to describe the contract between the service provider, his customer and a possible third party. These languages were defined closely to the common language allowing a common understanding

¹Part of the research described in this chapter has been published in [PVSAN⁺08, CANP⁺08]

of the service provider commitments to perform a service according to agreed guarantees. Several initiatives have been defined, and all of them are a complement to the service description implemented by WSDL.

One of the best known is Cremona [LDK04], a SLA middleware complementing the basic Web Services stack. Cremona helps providers to read, manage agreement templates, implement the agreement protocol, check availability of service capacity and monitor agreement states at runtime. Cremona's communication component handles various message types and message sequencing to form interaction protocols through the agent knowledge bases. In spite of that, the framework seems not to be competent on decision making based on the content of an agreement, job scheduling and resource management. Also, the monitoring components in Cremona do not provide support for agreement breaking. Finally, workflow handling, which is essential when dealing with complex interactions between services, is not supported.

The Web Service Level Agreement (WSLA) [LKD⁺03] is a framework targeted at defining and monitoring SLAs for Web Services [KL03]. The general structure of an SLA in WSLA includes the involved parties, the SLA parameters, the metrics and algorithms to compute those parameters, the service level objectives (SLOs) and the actions to be taken if a violation has been detected. The WSLA Framework implementation is based on the IBM Web Services Toolkit and licensed as commercial software. Its main functionalities are the definition, negotiation, deployment, monitoring, and enforcement of SLAs.

However, WSLA does not fully support multiple consumer/provider contracts, as these are signed by only two parties. Concerning partner responsibilities, it should be noted that defining them exclusively in terms of parameters and metrics is quite limiting, and there is no support for the definition of actions to be fulfilled. Finally, another very important issue is the lack of a generic, flexible and automatically executable mechanism for corrective management actions.

In the case of WS-Agreement [ACD⁺05], the general structure of agreements consists of the description of the context in which the agreement is established, the service itself and the guarantee terms. The WS-Agreement specification is less focused on the description of the related activities that should be choreographed but on the definition of the commitments and penalties.² WS-Agreement is quite often used with the conversation definition language WSCL [BBB⁺02]. The WSCL/WS-Agreement over IBM's ETTK is the reference (but partial) implementation of these specifications. It is built on top of Cremona and extends it by using WS-Agreement templates, richer message types, and XML-based interaction protocols. However, a WS-Agreement-based framework has quite some limitations: it does not include the specification of third parties working in the management of contracts, and no metrics are defined in order to support flexible monitoring implementations over the Web service choreographies.

PANDA [Bou08] is a project developing technology for the negotiation, monitoring and evaluation of contracts in supply-chains of producers in modular distributed Enterprise Resource Planning (ERP) systems. The infrastructure combines centralized Web service-based components (catalogue of partner profiles, SLA templates storage, etc.) with distributed peer-to-peer components implemented using JADE multi-agent platform. PANDA is an attempt to connect, at a high level, Service Level Agreements and Multi-

²WS-Agreement is the only specification of those under study that includes the explicit declaration of *penalties*, but they consist only of sets of actions.

Agent Systems, focusing on semi-automated negotiation and offline monitoring of contracts, and including interesting features such as matchmaking, negotiation, and Virtual Organisation (VO) evaluation. The main issues concerning PANDA for its use in a generic contracting architecture are that its main focus is in the domain of ERP-solutions, and that it is still under development.

Recently many event-driven (ECAs) Web standards have been developed, with particular emphasis on reactive RuleML languages and their SLAs-tailored variant, namely RBSLA (Rule-based Service Level Agreements) [Pas05]. However, RBSLA does not explicitly adopt the Web service-technology. Standard generic rule and inference engines such as Mandarax [Die05], based on RuleML [Bol06], and Prova [KS04] have been developed to execute and manage contracts designed in RBSLA. Prova, in specific, supports complex reaction rule-based workflows, rule-based complex event processing, distributed inference services, rule interchange, rule-based decision logic and dynamic access to external data sources, web-based services and Java APIs. Nevertheless, both frameworks have an object oriented aspect, concentrate on the support of inference and reasoning engines, do not provide direct support for contracting procedures between agents and operate on a declarative rather than deontic level.

Each of these existing pieces of work provide a different perspective on contracts and Web services, but still tackle the problem from a particular perspective such as language (RBSLA), negotiation (Cremona) or specification of the contract content (WSLA).

3.2 AN INTERMEDIATE STEP: CONTRACTUAL INSTITUTIONS

We have pointed out in Section 3.1 the flaws that the SLA in SOA initiatives have. The drawbacks describe have been the ones especially relevant for introducing an *e*-institution-based norm enforcement mechanism in SOA. However, the concept of SLA, if properly implemented, would be attractive for our purpose. As we noted in Section 1.4.2, institutions are created *by specifying* a set of norms that the members of the organization should fulfill.

Therefore, in principle we should be able to build institutions by using something more generic than SLA. In fact, SLAs are just specifications of *contracts*, however somewhat limited as we have seen.

A contract [SW03] is a written or spoken agreement that is intended to be enforceable. This agreement contains a set of clauses to be enforced. Each clause can be seen as a norm, so at the moment that it is signed by the parties, there is an institution created. This approach, as seen in Section 1.4.2 has already been studied by Dellarocas with the concept of Contractual Agent Societies [Del00].

We can then informally define a *contractual electronic institution* as an electronic institution which has been created at the time when an *electronic contract* has been signed by its parties. The members of the institution are these parties, and the norms to be enforced by the institution are the clauses of the contract.

In order to have a norm enforcement mechanism in SOA based on contractual electronic institutions, we will need a formalism to model contextual norms, in the form of electronic contracts, that avoids the problems SLA formalisms suffer from in this subject. Also, we will need a framework that allows for services that can create, manage, and enforce contracts, that is, for the creation of contractual electronic institutions.

Our research done in this field takes place towards these two directions, in the scope of the IST-CONTRACT project, where we propose a move to a more flexible contracting mechanism for Service Oriented Systems based on the following three main elements:

- The introduction of *intentional semantics* within the communication between services (based on the use of performatives such as request, inform, or commit). This is important as it re-enforces the link of actual and intended behavior.
- The creation of a *contracting language* able not only to express a set of intended behaviors on which parties agree but also to define the way that contracts and contract-related events are negotiated and communicated.
- The creation of *higher-level behavioral control mechanisms*, centered not on the tracking of a limited set of metric values but on the monitoring of higher-level objects such as commitments, obligations and violations which can be extracted from the communication semantics.

The combination of these three elements makes it possible to monitor the behavior of a set of actors by keeping track of the fulfillment of the agreements between them.

First of all, we present our contributions on an electronic language for contracts in Section 3.3, and our contributions on a middleware for creating contract-aware services in Section 3.4.

3.3 CONTRACTING LANGUAGE

This Section presents ongoing work in the definition of a *contracting language* which can be used not only to specify agreed behavior in service-oriented architectures but also for agent-mediated systems. The language is based on deontic notions such as obligations, permissions and prohibitions. The language not only covers the contract document itself but several layers of communication, including the messages and the protocols for contract handling.

In the next section we present an example introducing the domain problem while in section 3.3.1 we describe the conceptual layers of our contracting language. Section 3.3.2 analyses the important elements of the contract representation model. Section 3.3.3 explains how messaging and communication are achieved between the contracting parties.

For a detailed description of the formal semantics of this contracting language, please check [OPVSM08].

3.3.1 Layers/Elements of the Contracting Language

The contracting language separates concerns between different layers of communication:

1. The *Domain Ontology Layer* contains the domain ontology, providing ontological definitions of terms, predicates and actions (e.g. *car*, *workshop*, *repair*), to ensure that the same definitions are used by all parties and to avoid terminological misunderstanding.
2. The *Contract Layer* defines deontic statements about the parties' obligations, permissions and prohibitions in terms of predicates and actions defined in the previous layer (e.g. the workshop is *obliged to repair the car in 2 days*). The definition of such deontic statements is expressed as *clauses* in the contract.

3. The *Message Content Layer* defines the message content, allowing agents to express statements about contracts (e.g. a contract being *active/inactive*, *fulfilled*, *complete/incomplete*), actions related to contracts (e.g. *accept*, *sign*, *cancel* a contract) or expressions about actions and events related to the clauses in a contract.
4. The *Message Layer* allows agents to express their attitudes about the content of the message (e.g. an agent *proposes* to sign contract C1, or an agent *requests* cancellation of a contract C2). Taking Speech Act Theory as a basis, these attitudes are expressed by means of a standard set of pre-defined *performatives* which (similarly to FIPA ACL) are included as part of the message envelope.
5. The *Interaction Protocol Layer*, which pre-defines contract handling protocols as sequences of messages. These protocols structure interaction by defining sets of acceptable sequences of messages which would fulfil the goal state of the protocol (e.g. a protocol for agreeing on contract termination).
6. The *Context Layer* describes the interaction context where contractual parties will carry out the obligations, permissions and prohibitions agreed in the contract. This context also includes a model of the external regulations that may affect/rule the interaction between parties in the contract.

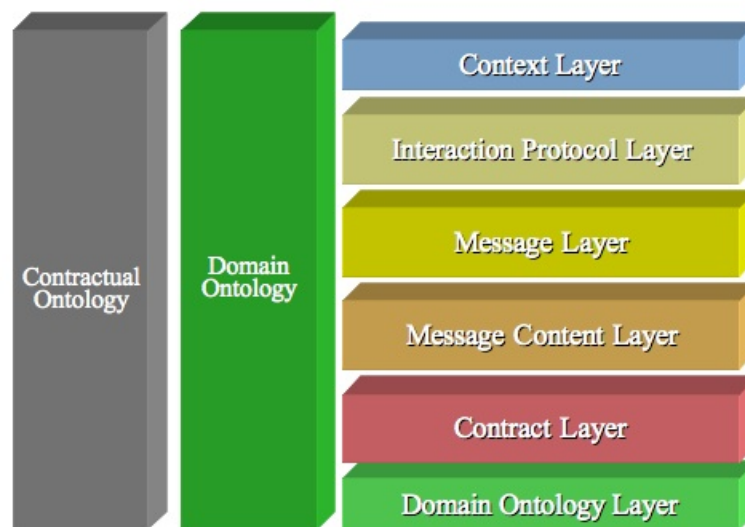


Figure 3.1: General view of the Contracting Framework

Apart from these horizontal layers, there is a vertical dimension that appears at all layers: the ontological dimension. This dimension defines all terms that are needed for each layer. In this approach there are basically two types of ontologies:

- **Domain ontologies:** these define the terms, actions, predicates and relationships needed for communication in a given domain. For instance, in the case of a car insurance application, there will be one or several ontologies defining terms such as *car*, *insurance*, *damage*, actions such as *repair* and predicates such as *repaired*.

The domain ontologies are defined in the domain ontology layer, but are also used in other layers such as the contract layer, the message layer or the context layer.

- The *Contractual Ontology*: an ontology that predefines all the terms, predicates and actions that are used by the framework, independently of the application domain. This ontology defines terms such as *contract*, *party*, *obligation*, *action*, *commitment* or *violation*, actions such as *creating* a contract, *committing* to a contract and predicates such as *fulfilled* or *cancelled*.

3.3.2 Contract representation

The Contract Data Model is an XML based representation suitable to represent both fully defined contracts or contract templates (contracts partially defined). It is based on the theoretical framework defined in [PANVS⁺08]. It is composed by a name, starting and ending dates and three main parts: contextualization, definitions and clauses. Figure 3.2 depicts this structure.

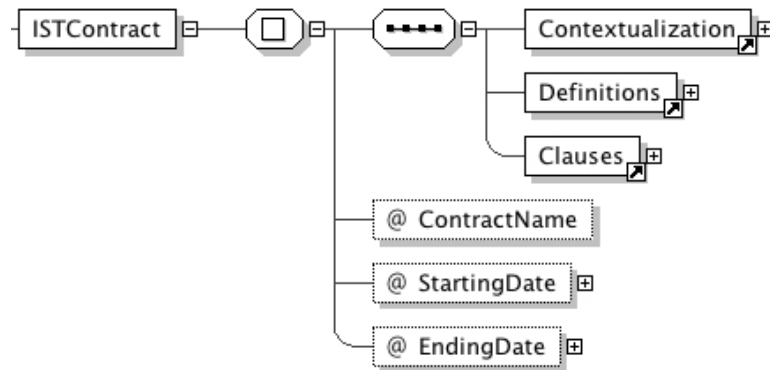


Figure 3.2: Root elements of the Contract representation

The name of a contract has to be unique inside the context it is being declared in (a contract is uniquely identified by a combination of the context namespace and the contract name). The starting and ending dates of the contract express the valid time period of the contract. Ending date is optional (contracts without end date are allowed by leaving the ending date blank). The rest of the elements are described in the following sections.

Definitions

The *definitions* part defines the parties of the contract, the roles they play, some optional grouping of roles and the model of the domain.

Parties. The *contract parties* are the the list of agents involved in the contract, that is, the set of agents assigned to fulfil one or more clauses of the contract. In the contracting language, the contract parties element has for each agent its name, a reference to this agent and optionally a text description about this agent (see Figure 3.3).

Role Enactment List. The *role enactment list* element is used to assign roles to the agents (parties). The definition of roles is not common in existing contracting languages (such as WS-Agreement or WSLA), which tend to use directly some kind of agent identifier to assign responsibilities. In our approach roles are a very powerful mechanism that decouples

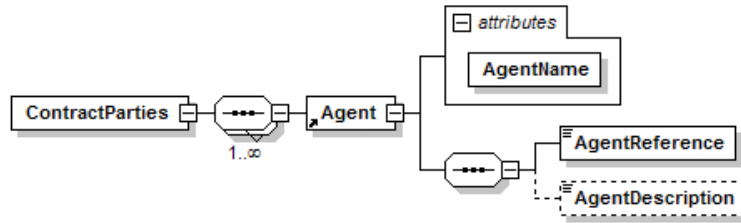


Figure 3.3: Parties element in contract

the definition of responsibilities from the specific agents that will have to fulfill them. This decoupling allows to create *contract templates* which fully specify, e.g., the obligations of a repair company or a insurance company in an archetypical repair scenario without specifying the exact agents enacting the roles. Such contract template can be then instantiated several times by only specifying each time the exact parties and the role-enactment relations.

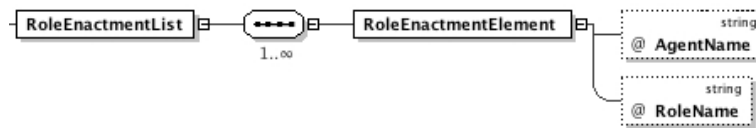


Figure 3.4: Role enactment list element in contract

Figure 3.4 shows the structure of a Role-enactment list.

Group List. The *group list* element is used to group agents that have different roles into a group that might share some responsibilities. It can be useful when there are clauses that should affect a subgroup of the agents in the contract. Each group of the list has a name and the list of agents that compose the group (see Figure 3.5).

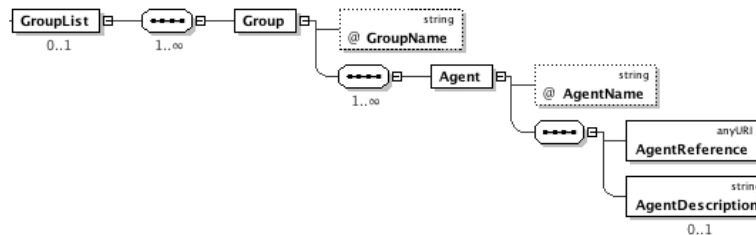


Figure 3.5: Representation of the Group list element

There is always a predefined group called *ALL*, which groups all agents in a contract. Also all roles implicitly create a group of all agents that enact the same role. Therefore it is not needed to create a group of agents which enact the same role.

World Model. Within the contracting language, it must be ensured that all parties have a shared understanding of the elements in the world that they will refer to in their interactions and also of the characteristics of the world itself. A representation of the different elements composing the knowledge about the domain is depicted in Figure 3.6.

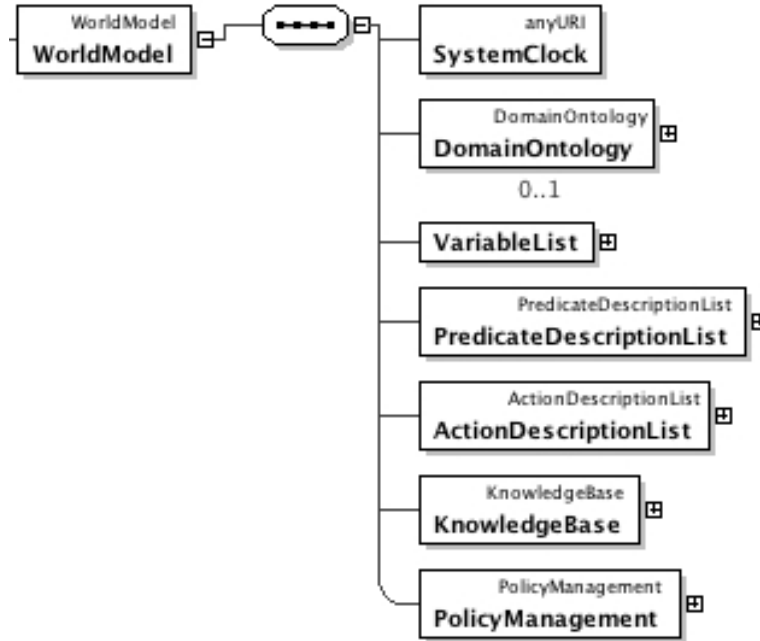


Figure 3.6: Representation of the world model element

Contextualisation

In the contracting language, contexts are implicitly created when a contract becomes active. They will obtain elements from the contract, such as the world model, the domain ontology, the descriptions of possible actions and processes within the domain and the set of regulations to be applied within the organization being represented. Moreover, the context of a contract, called *interaction context*, may be as well contained inside another interaction context and therefore inherit all its knowledge representation elements and be constrained by its regulations. In this case, the *Contextualization* part of the contract has to specify which is the *parent contract*. If a contract is an instance of a *contract template*, this is also specified in this part.

This concept of interaction context is equivalent to the *Context* defined in HARMONIA[VS04].

Clauses

Clauses express agreements between parties in the form of deontic statements. In order to express the clauses we have adopted a variation of the representation defined in [Ald07], which is based on a dyadic deontic logic including conditional and temporal aspects.

It is important to note here, that although the name given here is *clause*, which is typically used in the definition of a contract document, we are in fact representing *norms* in a *norm condition expression language* [VSAD04].

A clause (Figure 3.7) is structured in two parts: the conditions and the deontic statement. There are three conditions, which have the form of boolean expressions, that have to be evaluated at different stages of the clause life cycle.

- *Activating (or triggering) Condition*: when this condition holds true, the clause is considered to be activated. This condition can also be referred to as precondition. If the boolean expression of this condition includes a `violated()` predicate, the clause is considered to be a violation handler.
- *Exploration (or maintenance) Condition*: this is the invariant of the deontic statement execution, which means that when the clause is active and the statement is being enforced, the exploration condition has to hold always true. If this does not happen, a `violated()` predicate will be raised. This condition can include temporal operators *before()* and *after()* with *allow* to express temporal constraints and deadlines.
- *End (or achievement) Condition*: the clause is considered to be inactive and successfully fulfilled if and only if the exploration condition has always held true and the end condition holds true.

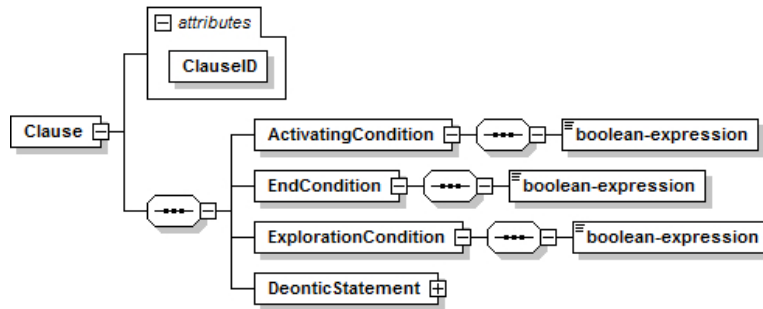


Figure 3.7: Representation of the clause element

The *DeonticStatement* is the central element of the clause. There are three main fields in its structure: the deontic modality, the roles involved and the object of the norm (Figure 3.8):

- *Modality*: this field indicates the deontic modality of the statement, which can be either Obligation, Prohibition, or Permission.
- *Who*: contains the set of roles and groups of roles that will have to fulfil the statement.
- *What*: this represents the object of the norm. This object can be an action or a state. If it is an action, this action will have to be executed in order to fulfil the norm. Otherwise, if it is a state, the responsible actor(s), defined in the *Who* attribute, will have to ensure that this state is accomplished as long as the exploration condition holds true.

Clauses can be used in two ways:

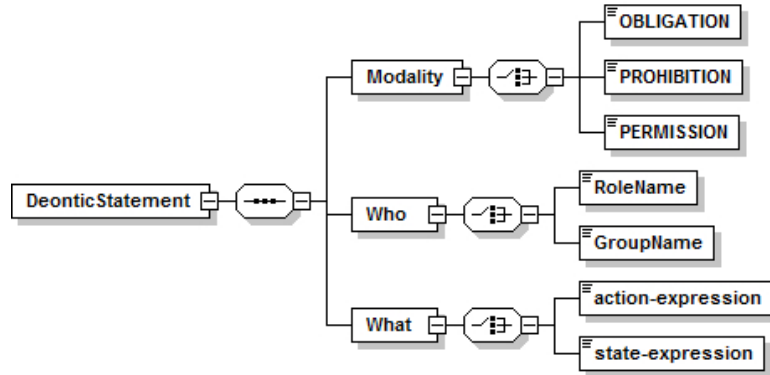


Figure 3.8: Representation of the deontic statement element

- as *standard clauses*: the ones that define what ought/ought not to be done. For instance, a clause stating that a buyer should pay for a given item within some time period.
- as *violation handling clauses*: the ones defining what to do if standard clauses are violated) (i.e. The exploration condition does not hold).

An example of a standard clause could be, if we suppose a car insurance scenario, 'The repair company is obliged to notify Damage Secure before april 10 if the report of the repair is ready'. Such clause, which in dyadic deontic logic would be formalised as:

$$O_{RepairCompany}(exists(RepairReport(car)) \rightarrow sendRepairCompleted(car, DamageSecure, RepairCompany) \leq April10)$$

can be expressed in our language as follows:

```
<Clause ClauseID="NotifyRepairCompleted">
  <ActivatingCondition>
    <BooleanExpression>
      exists(RepairReport, isRepaired(car12f3pw, R1))
    </BooleanExpression>
  </ActivatingCondition>
  <EndCondition>
    <BooleanExpression>
      isSentRepairCompleted(car12f3pw,
        RepairReport, "Damage Secure", R1)
    </BooleanExpression>
  </EndCondition>
  <ExplorationCondition>
    <BooleanExpression>
      Before(2008-04-10T15:30:30+01:00)
    </BooleanExpression>
  </ExplorationCondition>
  <DeonticStatement>
    <Modality>
      <OBLIGATION/>
    </Modality>
    <Who>
      <One id="R1" enacting="Repair Company"/>
    </Who>
  </DeonticStatement>
</Clause>
```

```

</Who>
<What>
  <ActionExpression>
    sendRepairCompleted(car12f3pw,
      RepairReport, "Damage Secure", R1)
  </ActionExpression>
</What>
</DeonticStatement>
</Clause>

```

This clause only activates when a report stating that the car is repaired exists, and deactivates once the report is sent. The *ExplorationCondition* specifies in its case that the obligation should be met before a given deadline. The *DeonticStatement* specifies that this is an obligation related to one agent, R1,³ enacting the *Repair Company* role, and it consists of one action (sending the repair report).

Contractual Ontology (IST-Contract)

As mentioned in Section 3.3.1, the *Contractual Ontology* is the ontology that predefines all the terms, predicates and actions that are used by the framework, independently of the application domain. The purpose of this is for a generic, non-application dependent information to exist within the framework and be shared amongst the agents.

This ontology defines objects which represent primal entities existing within the framework, actions committed by an actor and predicates which declare states and conditions of the system.

Some of the primal concepts that exist in the contractual ontology are:

- Contract: an agreement between several parties
- Obligation: an obligation corresponding to an agent
- Party: a person, agent or entity
- Action: an action taken by one of the parties
- Predicate: a logical predicate with zero or more arguments which is true or false
- List: the classical notion of list
- Penalty: the penalty that one party receives in case of a violation of an obligation

The table below shows two examples of pre-defined actions and predicates to express statements about contracts are shown. The first is the action of committing to a contract and the second is the action of creating a contract. Attributes represent both inputs and outputs. Wherever a predicate appears in the precondition and does not in the postcondition, it is assumed that it remains the same after the action is completed.

Other actions, defined in a similar way are: terminate, withdraw, cancel, get, update and more.

Predicates are of the form of:

- committed(P, C), where P is instance of a party and C is instance of a Contract

Other predicates, defined in a similar way are: happened, all-committed, violated, obliged, initiated, active, cancelled, and more.

³It is important to note here that we show a version of the clause that is valid for both contract templates and contracts, where R1 is a variable which value is set by unification in the activating condition (i.e. the Repair Company that created the Repair Report). If one is not interested to reuse this clause in several contracts, R1 could be directly substituted here by the specific Repair Company.

<i>action</i>	<i>attributes</i>	<i>precond.</i>	<i>postcond.</i>
commit	P: Instance of Party C: Instance of Contract	C is valid	committed(P, C) initiated(C) \forall O Instance of Obligation in C: obliged(P, C, O) happened(P, commit(P, C))
create	P: Instance of Party C: Instance of Contract	\neg (exists(C))	exists(C) happened(P, create(P, C))

Table 3.1: Action of committing a contract and action of creating a contract.

Domain Ontology

Domain ontologies are important elements in our framework, as they semantically connect references to the same entities and objects made at different layers (e.g. a message referring to a car and a contract referring to the same car). These ontologies also ensure that the same definition for actions and predicates is consistently used, avoiding interpretation errors. This last aspect is especially relevant in contracts, as all parties should have the same understanding about the agreed terms in the contract. For instance, if the contract establishes the obligation of a given repair company to fix the client’s car windshield, both the repair company and the client should have the same definition for the concept “windshield” and the action “to fix” when applied to windshields.

There are at least two levels of domain ontologies in the CONTRACT Communication Model: contract ontologies and context ontologies. More levels may appear if super-contexts of a given context are defined which include ontological extensions, or if sub-contracts in the scope of a given contract are defined which include additional concepts.

Context ontologies define all shared definitions by actors interacting in the domain. These ontologies will be mostly classes, without instances. Instances will only be included in the context ontologies if they are to be re-used in more than one contract and if the number is not too big.

Although the context ontologies are not part of the data model, there are some requirements that they should fulfil. For instance, they should define all predicates and actions (with their parameters), roles and terms which appear in the rest of the context definition.

Contract ontologies are extensions of a given context ontology (or an overarching contract) which inherit, refine and/or extend the terms needed for that particular contract in a given domain. These ontologies also define instances of existing ontological classes, when enumeration of all instances is not advisable at the context ontology. In those cases where all needed classes and instances are already defined in a context ontology, then no extension should be defined and the contract ontology and the context ontology are the same.

The model thus depends on the capability to extend existing ontologies with additional concepts and relations. The extended ontology should contain not only the added terms, but also inherit all definitions from the extended ontology.

Domain ontologies are explicitly used in several parts of our model. Communication messages include an explicit reference to the ontology needed to interpret the content of

the message (Section 3.3.3). Contracts also include, in their World Model part, a reference to a domain ontology.

If no additional ontology is defined, then the ontology used is the context ontology (Section 3.3.2).

In general, all elements and concepts used in the message content, in contracts and definitions of context should correspond to classes and/or instances which should exist in some ontology.

3.3.3 Contracting Messages and Protocols

In order to increase the expressivity of the communication between services to introduce some intentional stance, the contracting language also defines a set of performatives to be used by the parties. We have extended FIPA ACL [fIPA02] performative set in a way that can be used not only by web services but also by agents. To properly use those performatives, our contracting language also specifies the message structure, a content language and a set of contracting protocols

Message Structure

FIPA [fIPA00] provides both a message format and a message-handling protocol to support run-time knowledge sharing among agents. It can be thought of as consisting of three layers: a protocol (performative) layer, a content layer, and an ontology layer. The domain-independent performatives in protocol layer, such as inform, propose, accept-proposal and agree, describe the communication actions between agents. FIPA-SL defines the syntax and semantics for the expression of communication content.

Our proposed message structure is an XML variation of FIPA's message structure [fIPA02], where the message body contains the usual FIPA proposed attributes, i.e. sender, receiver, performative, language, content, etc.

Message Content

The content of the communication message describes what the purpose of the communication is and expresses knowledge existing in the level of the world representation.

That is, the ontology and its elements are used to support the interpretation of the message content by the receiving agent and they can be found in 3.3.2 and 3.3.2 as part of the context ontology.

One important feature of the contracting language is that contracts can be part of the content of a message. This is possible due to this concept being defined in the contractual ontology, which will be a basic domain framework for the communication, and provides a flexible way to express knowledge related to specific contracts.

The language used to express the content of the messages between the actors is based on a subset of FIPA-SL [fIPA00]. FIPA-SL is more complete than other logic languages, such as Prolog, when focusing on content-oriented agent communication, and at the same time it is more complete than other semantic languages, as KIF [FFMM94], allowing for better constructs with better semantics and lower complexity.

A subset of FIPA-SL, namely FIPA-SL2, adapted to an RDF representation⁴, has been chosen as a basis for the content language. The reason for this is that it remains an adequately expressive set as it allows first order predicate, modal logic operators, quantifiers

⁴RDF has been chosen here instead of XML because it is easier to integrate with semantic representations such as OWL.

(*forall, exists*) and reference operators (*iota, any, all*, which are needed in order to give the expressivity needed for flexible contract-related communication.

Performatives

The full set of FIPA-ACL [fIPA02] *performatives* is adopted (e.g. *query, inform*, etc.). However, FIPA standards do not include cases in which an agent desires to propose an action to be performed by more than one agent (e.g. to propose that many agents commit on a contract). For this reason, FIPA-ACL alone is not sufficient, as its speech acts cannot be used to form, maintain and dissolve joint intention (mainly to commit) in order to support advanced social activity (i.e., teamwork).

We have extended FIPA-ACL performatives with extra performatives based on joint intention theory [Tuo96]:

- **suggest:** The action of submitting a suggestion for the sender and the receiver agents to perform a certain action.
 $\langle i, \text{suggest}(j, \langle i, \text{act} \rangle, \langle j, \text{act} \rangle) \rangle$
 where *i* is the sender and *j* is the receiver.
- **consent-suggestion:** The action of showing consent to a suggestion for the sender and the receiver agents to perform a certain action.
 $\langle i, \text{consent-suggestion}(j, \langle i, \text{act} \rangle, \langle j, \text{act} \rangle) \rangle$
 Agent *i* informs *j* that, it consents for agent *i* and agent *j* to perform action *act* giving the conditions on the agreement.
- **dismiss-suggestion:** The action of dismissing a suggestion for the sender and the receiver agents to perform a certain action.
 $\langle i, \text{dismiss-suggestion}(j, \langle i, \text{act} \rangle, \langle j, \text{act} \rangle, \psi) \rangle$
 Agent *i* informs *j* that, because of proposition ψ , *i* does not have the intention for *i* and *j* to perform action *act*.

Protocols

The architecture defined in [PANVS⁺08] identifies agent behaviors which should be well defined and designed, in order for the contract lifecycle to be smoothly executed. These include the phases of Contract Creation, Contract Fulfilment, Contract Modification and Update, Contract Violation, Contract Cancelling By Agreement and more. Such a communication can be achieved through communication *protocols* which define significant part of every agent's behavior.

A set of contract handling protocols have been created to support those behaviors. Figure 3.9 depicts an example of a protocol expressing the creation of a contract between two agents.

Protocol handlers have been developed for both agents and agentified web services, easing the implementation of the communication to the designer.

3.4 CONTRACTUAL SERVICE MIDDLEWARE

This Section presents a *middleware* to help designers in the implementation of *contract-aware agent-based services*, matching a formal model of an overall contracting process and environment in a general way. This middleware provides several components, including a contract manager, a communication manager and a workflow manager, which combine to allow agents to manage contracts and the actions associated with them.

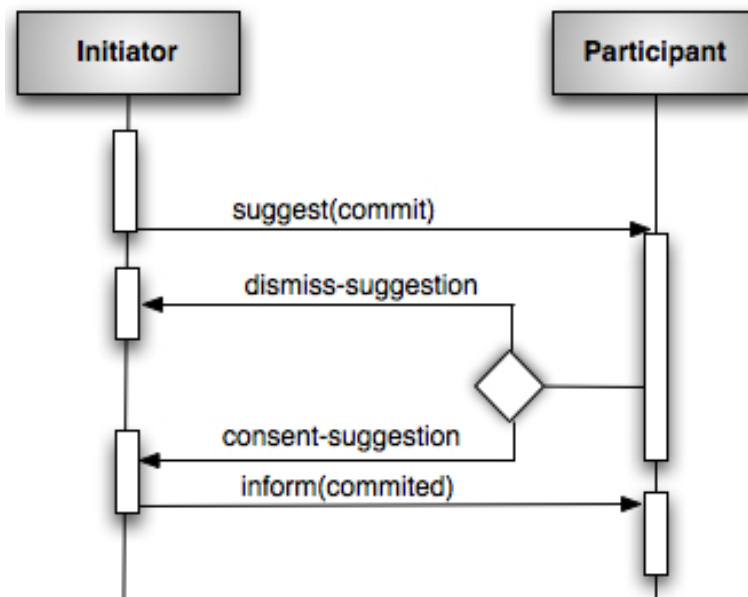


Figure 3.9: Simple Contract Create protocol

With this middleware we can create contractual environments, in which services can:

- Create contracts.
- Handle all contract-related communication.
- Manage the active responsibilities during the contract execution.
- Solve disputes if third, administrative parties are included in the system.

Giving these capabilities to the services, we are in fact providing a mechanism with which to *create* and manage *electronic contractual institutions*. As we have seen in the beginning of this Chapter, this an important piece to manage institutional norm enforcement in SOA.

This middleware is being built on top of a contracting environment [WBC⁺08] which is a Web service implementation of the IST-CONTRACT framework [OML⁺08]. The framework and the agents can be instantiated in a *contracting deployment* when delivering real business applications. This leads to two main issues: the creation of the *contracting environment* the agents operate in, and the creation of the *agent middleware service architecture*, a set of components easing the creation of contract-aware agent-oriented services. As internal components communicate through Java APIs, designers can decide to plug-in their own developed modules.

The middleware is now being tested in three case scenarios in the context of the CONTRACT project. Current version uses a simple reasoning cycle in the Decision Maker module implemented in 2APL. One of the next steps is to try to adapt other intelligent agents technologies, such as Drools or JADEx.

In the following sections we cover aspects regarding, mainly, the internal components of the middleware.

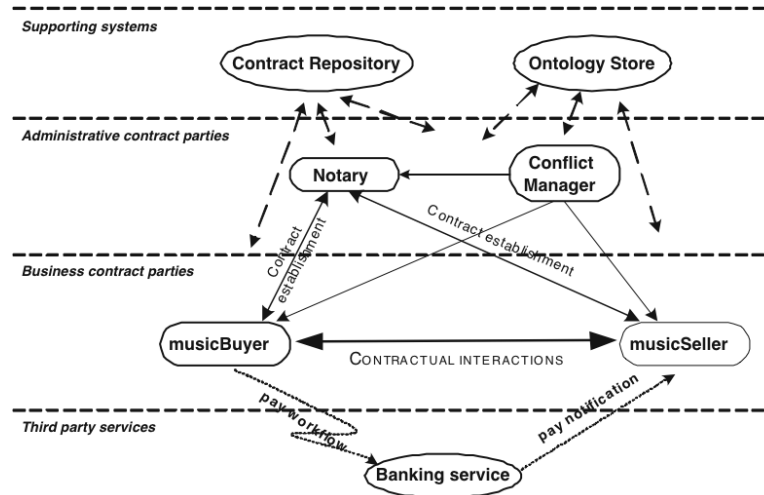


Figure 3.10: The contracting environment in the example *buyMusic* scenario

3.4.1 Contracting Environment

The *contracting environment* is an instantiation of the theoretical framework involved in contract-related processes. Each agent may enact one or more roles (e.g. a *business contract party* or both an *observer* and *manager* for example), including administrative roles as agreed by the parties to a contract. Figure 3.10 shows an example of two business contract parties, a *musicBuyer* and a *musicSeller*. Administrative contract parties' roles are all enacted by extra agents: a *Notary*, and a *Conflict Manager*, enacting the manager and the observer roles in the framework. The figure also shows how the contracting environment provides additional support systems such as a *Contract Repository*, an *Ontology Store* which ease the development of distributed contract-based applications. Additional supporting systems such as directory service (acting as agents yellow page) and context service (providing interaction context rules) can be added to the environment as well.

In the following we present our proposal of the intended agent service architecture for building contract-aware agent-based services.

3.4.2 Administrative contract parties

As we saw in Section 1.4.1, the members of a society use the institutional constraints in order to reduce the cost of the interaction between each other. The events and assertions related to these interaction constraints thus need to be observed and enforced by the members themselves *or* with the help of third parties.

Administrative contract parties represent these third parties. They provide administrative support for a contract during its lifecycle. In the CONTRACT framework [OML⁺08], the explicit roles specified for contract administration are the Observer and the Notary.

These roles are implemented following the internal architecture described in Section 3.4.3.

Observer

The *observer* can notify listeners, when an obligation is not being fulfilled or in danger of not being fulfilled. By observing only those states and actions that are applicable to contract execution, information exposure is minimized. The observer plays a critical role in contract management, observing the state of the system and notifying agents and components of state changes when appropriate.

The observer allows for the abstraction of domain specific observation requirements.

Manager

A *manager* knows about a breach of contract by registering to listen to the notifications from an observer. Manager is a role, and one agent may play the role of both manager and observer.

The nature of the action taken by a manager may vary considerably. In highly automated and strict applications, an automatic sanction may be taken from a party. In other cases, a management agent may be a person that decides how to resolve the problem. Alternatively, a manager may merely provide analysis of problems over a long term, so that a report can be presented detailing which obligations were missed.

A manager can be, or act with, a third-party arbiter, independent from the managed contract's other parties.

3.4.3 Internal architecture

Our proposal for the *internal architecture* of the agent service middleware is shown in Figure 3.11. Different public interfaces allow contract-aware agents to communicate with each other and with other components of the contracting environment. Public interfaces are of three types:

- *agent-to-agent*: is a Web service interface which allows agents to communicate at the knowledge level according to the contracting language and protocols defined in Section 3.3. FIPA-ACL performatives are exchanged as Web service messages using SOAP.
- *agent-to-system*: is a Web service interface used for communication between the agent and supporting systems such as the Ontology Store and the Contract Repository.
- *agent-to-user*: is a Web interface through which the contract-aware agent interacts with system users (human interface) and with the real world. The former facilitates interactivity with the user, the latter interoperability with third party systems (such as Merchant provider for the on-line payment).

Architecturally, a contract-aware agent is split into several modules which provide specific functionalities and knowledge. The modules communicate through Java APIs.

Decision Maker

The *decision maker* contains the core intelligence and main reasoning cycle of the agent, but the implementation is left to the designer. This would allow to program the agent with contract-related behaviors, which model the deliberation and the achievement about contract clauses.

To implement contract-related behaviors, the decision maker is assisted by the *contract manager*, the *communication manager* and *workflow manager*. These modules not only handle

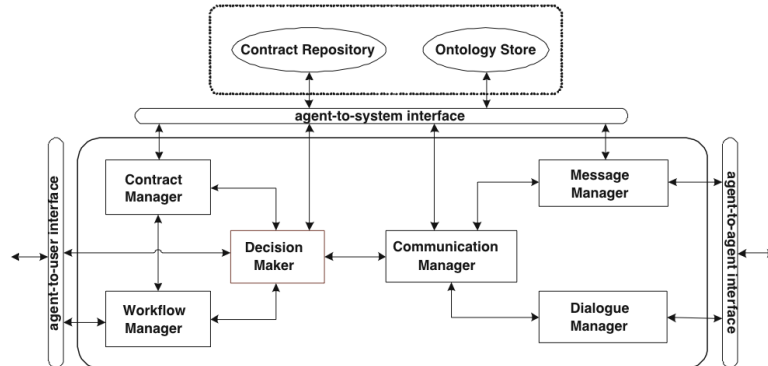


Figure 3.11: Agent Service Middleware Architecture

the low level details of messages, contract agreements and workflows but also keep an up-to-date model of the status of all these aspects which can be easily queried by the decision maker.

This separation of concerns eases the implementation from the designer perspective, as the reasoning cycle inside the decision maker focuses on strategic decision-making.

Contract Manager

The *contract manager* contains the contract knowledge and the business logic of the contracts in terms of predicates and actions (which are formally defined in the *Ontology Store*).

The contract manager is aware of the contract deontic clauses that apply to the given agent, their status (active, inactive, violated), and the overall status of the contract. The contract manager also keeps track of pending obligations and signals the decision maker module about next obligations to be achieved, risks of violating a clause or the fulfillment of the contract.

Workflow Manager

The *workflow manager* contains the operational knowledge required to execute and monitor the contract workflow, i.e. the order in which the actions associated to the deontic clauses of the contract have to be carried on. Operational knowledge is expressed in terms of action's inputs, outputs, and pre- and post-conditions. Pre- and post-conditions straightforwardly map to activating and terminating conditions of contract deontic clauses.

In this way, a workflow execution results in the execution of a sequence of actions which eventually satisfy contract obligations. Fulfilled sequences are signalled to the contract manager, which marks active obligations as fulfilled and forwards the status change to the decision maker module, which can deliberate then on the next step to do.

Communication Manager

The *communication manager* handles all issues related to agent-to-agent communication. One of its functionalities is to act as a local Directory Facilitator for the business agent, being aware of the network topology, i.e. it knows which are the active agents in the system and how to contact them.

Secondly it knows which interaction protocols to choose for contract-related behaviors. E.g., during the process of setting a new contract, the communication manager is queried by the decision maker about the protocol to use to achieve a communication goal (setting the contract).

The communication manager is assisted by two modules: the *dialogue* and *message managers*.

Dialogue Manager

The *dialogue manager* implements a library of interaction protocols, storing the sets of acceptable sequences of messages which fulfill the goal state of a protocol for each of them.

The dialogue manager is implemented as a finite state machine, in order to keep track of the current state of the dialogue and ensures that interactions are consistent with respect to the dialogue structure.

Message Manager

The *message manager* processes the content of messages received from and sent to other agents.

Regarding the incoming messages, it is responsible for the semantic interpretation of their content, translating it into an RDF representation that can be queried as a knowledge base by the decision maker through the communication manager (e.g. the Decision Manager can be asked if any agent has sent a notification about a payment being done).

For outgoing messages, the message manager is responsible for the conversion of the agent's internal representation into the common, understandable format, used by all the agents. Translation rules can be defined according to ontological relations.

3.4.4 Supporting Systems

At least two *supporting system components* are needed by contract-aware agents: a *contract repository* and an *ontology store*.

The Contract Repository provides persistent storage of *contract templates* and *contract instances* avoiding the loss of information about contracts between sessions.

The Ontology Store is a repository for storing and retrieving domain and contractual knowledge, which provide ontological definition of terms, predicates and actions which are referred to in the terms of the contracting language.

These components can be deployed either only in the contracting environment or inside each contract-aware agent extending the core set of components previously described.

3.4.5 Example of Contract Management and Execution

In this section we describe how business and administrative contract parties are used in the design of a simple contract-based application, an on-line music store. We further illustrate how the internal modules we propose can be used. The scenario is depicted in Figure 3.10.

Business agents, such as the *musicBuyer* and *musicSeller*, are the signatory parties of the *buyMusicContract* and are involved in contractual interactions.

The *Notary* and the *Conflict Manager* are administrative contract parties which respectively support the establishment and monitoring of the *buyMusicContract*.

The Notary enacts the manager role (see Section 3.4.2) as a certification entity which supervises the agreement between *musicBuyer* and *musicSeller*. This administrative party is fed by the information provided by the contract manager, the communication manager

and the workflow manager in order to supervise the agreement between the parties and enforce the clauses of the contract⁵.

The Conflict Manager enacts the role of the observer (see Section 3.4.2), and monitors contractual interactions. It is responsible of detecting any conflicts that may arise between the *musicBuyer* and *musicSeller* during contract execution. The Notary is subscribed to the Conflict Manager in order to receive notifications regarding these conflicts.

The Contract Repository and Ontology Store respectively provide contract storage functionalities and ontological definition to the other agents. Finally, the *Banking Service* is a third party service involved in the *musicBuyer* payment workflow. It also notifies the *musicSeller* about the outcome of the payment.

According to the contracting language defined in Section 3.3, a contract document specifies signatory parties and the deontic clauses parties are subject to. In this scenario, the *musicBuyer* and *musicSeller* agents represent the signatory parties. For this example, the *buyMusicContract* includes six clauses:

- dc1.** if available(DownloadService) \longrightarrow
 PERMITTED_{*musicBuyer*}(buyDownloadRights(CD.Id))
- dc2.** if done(buyDownloadRights(CD.Id)) \longrightarrow OBLIGED_{*musicBuyer*}(pay(CD.Price)) BEFORE 1 Hour
- dc3.** if violated(dc2) \longrightarrow PERMITTED_{*musicSeller*}(abort(buyMusicContract))
- dc4.** if done(pay(CD.Price)) and (available(DownloadService) \longrightarrow
 OBLIGED_{*musicSeller*}(giveDownloadRights(CD.Id,musicBuyer)) BEFORE 1 Hours
- dc5.** if done(giveDownloadRights(CD.Id,musicBuyer)) \longrightarrow
 PERMITTED_{*musicBuyer*}(download(CD.Id) BEFORE 1 Hour)
- dc6.** if violated(dc5) \longrightarrow OBLIGED_{*musicSeller*}(payDelivery(Delivery.fee) and (deliverCD(CD.Id) BEFORE 7 Days))

Clauses are associated with a well-defined lifecycle. When a contract is agreed, clauses are *inactive*; they move into an *active* state when their associated activating conditions hold. Then, they may change to a *fulfilled* state when their associated actions are executed or to a *violation* state when their associated actions fail. Clause state changes are perceived through the message manager interpretation process of agents' messages. These percepts are forwarded to the contract manager (through the communication and decision maker modules), who changes the state of the clauses accordingly.

The run-time management of contract-bounded interactions goes as follows. A new instance of the *buyMusicContract* is created from a contract template whenever the *musicBuyer* starts the process of buying music on the online *musicStore*, managed by the *musicSeller*. At this stage, the Communication Manager, the Dialog Manager (and Message Manager) are involved in the contract establishment process. The Communication Manager provides the *musicBuyer* with details such as the IP address and port number for contacting the *musicSeller* and the protocol to be adopted (or the opposite, depending on which one of the two is the initiator). A Simple Contract Create Protocol can be used (see [PANVS⁺08] for details on this protocol) to negotiate and instantiate a new contract and the Dialog Manager makes sure that the protocol is followed correctly.

⁵Due to the lack of central platform in typical SOA architectures, the contract manager, communication manager and workflow manager are the way some of the institutional enforcement tasks are distributed among the actors in the system, creating a sort of distributed institutional platform.

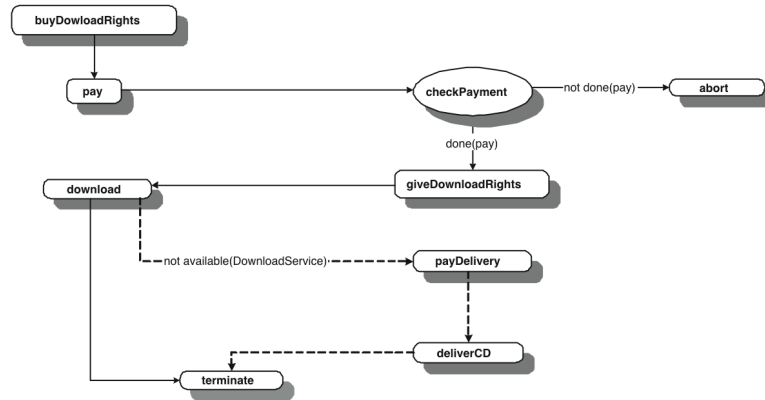


Figure 3.12: The workflow derived from the *buyMusicContract* clauses. During the *download* an exception may occur which triggers the activation of violation clause dc5 and the dotted path of the workflow is followed.

Once the *buyMusicContract* has been settled, the Decision Maker module extracts the deontic clauses the agent is responsible for. This knowledge about the status of the contract and its clauses, is kept by the Contract Manager. Agents are now ready to execute the signed contract. It is important to clarify here that clauses of a contract implicitly define a (partial) ordering in the sequence of the actions to be performed when executing the contract, i.e. the workflow (of the process described in the contract). The derived workflow of the *buyMusicContract* is shown in Figure 3.12.

The first action to be executed is the *buyDownloadRights*. According to clause dc1, download rights can be bought if the service is available. The Communication Manager converts the *musicBuyer's* intention to buy in a communication goal in terms of a request protocol (Figure 3.13(a)). The translation of the *musicBuyer* action representation into a understandable representation for the *musicSeller* is done by the Message Manager. In this case, the *buyDownloadRights* action is translated in a `REQUEST(sellDownloadRights(CD.Id))` message. The request protocol is started by the Dialogue Manager. When an agree message is received, the Workflow Manager will be responsible of carrying on the agreed action. The reception of an `INFORM-DONE` message will trigger the fulfillment of clause dc1 and the activation of clause dc2 as its activating condition holds. The *musicBuyer* and *musicSeller* interactions for the buy action and their clause states are shown in Figure 3.13(a).

Next, the buyer is obliged to pay for the download. To accomplish that, the *musicBuyer* interacts with a banking service according to a pre-defined payment workflow. Once finished, he acknowledges the *musicSeller* by an `INFORM` message (again, Communication Manager, Message Manager and Dialogue Manager are involved here). When the *musicBuyer* receives the payment confirmation from the banking service as well, it informs the *musicSeller* that the payment was successfully concluded, which will update the clauses' states (Figure 3.13(b)). It may however happen that buyer or seller have not matching perceptions about the payment. In this case, the conflict is detected by the Conflict Manager and notified to its subscribers. In this case, the Notary processes this conflict and therefore enforces the activation of violation clause dc3, ensuring that subsequent clauses become

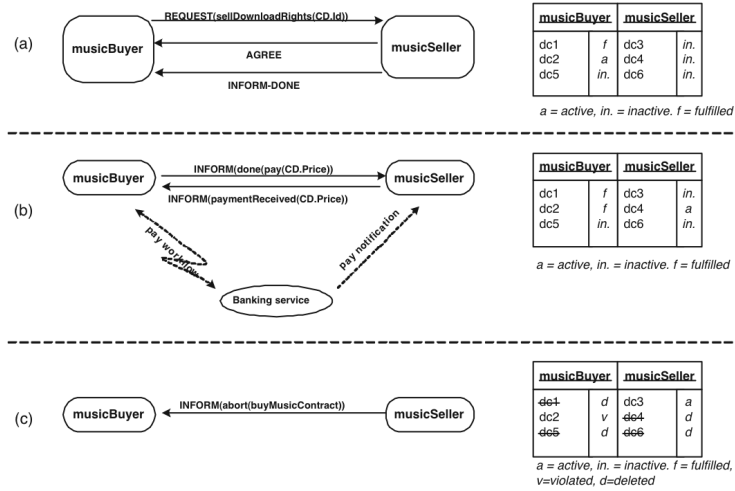


Figure 3.13: The *musicBuyer* and *musicSeller* interactions and internal states regarding the contract clauses for the *buyDownloadRights* action (a), *pay* action (b) and the *pay* clause violation (c) respectively.

invalid and previously fulfilled clauses are deleted. The seller then is permitted to abort the contract (Figure 3.13(c)).

When the payment succeeds, the *musicSeller* clause dc4 becomes active and so does the obligation to give the rights for the download. When dc4 is fulfilled, dc5 becomes active and the buyer is allowed to download the CD. In the case the download service becomes unavailable because of overloading of the server during the time period clause dc5 is still active, an exception is raised and the seller is responsible to deliver the CD before 7 days and to pay the delivery fees. Whenever all contract clauses are fulfilled, the agents can start a Contract Termination Protocol (see [PANVS⁺08] for protocol's details). When the protocol ends, the contract can be considered successfully terminated.

3.5 CONCLUSIONS

This Chapter has introduced a contracting language and middleware that suits our purpose in respect of managing norm enforcement in SOA.

In Section 3.1 we have summarized the state of the art of SLA in SOA, which is the current mechanism used by the SOA community for the enforcement of agreements. However, as we have seen, these proposals are not generic or suitable enough to be integrated in the HARMONIA framework.

Section 3.2 presents a workaround: the use of contracts, which are a more abstract specification of an agreement. With the use of electronic contracts, we can create electronic contractual institutions.

Part of our research has been done in electronic contracts. The language chosen for the definition of these contracts is defined in Section 3.3, while Section 3.4 contains a middleware for the management of electronic contracts in SOA. These two elements fill some gaps in our way to *e*-institutions in SOA.

In the first place, the language proposed is a formalism that represents a contract containing a set of norms (clauses) that fit a high level definition in deontic logic as seen in Section 1.4.1. This language is especially designed to be used in SOA and solves issues found in the various proposals of SLA for SOA.

The middleware described uses this language in order to provide an infrastructure on top of SOA to create, interpret, manage, and fulfill electronic contracts. From our point of view, it is a mechanism that can be used by services in order to create electronic contractual institutions. Therefore, this middleware is a solid option for being used as the platform in which to implement the proposal we will present in Section 4.3.

Thesis Proposal

This chapter describes in detail the PhD proposal introduced in 1.5. To do so we need some background about the HARMONIA framework (in Section 4.1) and the architecture of a SOA governance system (in Section 4.2), in order to be able to find common concepts of both definitions and make a matching between them in the proposal (in Section 4.3)¹.

4.1 HARMONIA

As seen in Section 1.4.2, HARMONIA is the institutional framework that best fits our purpose of modelling electronic institutions, thanks to its multi-level approach.

HARMONIA framework [VS04, VSD03] is a multi-level approach to model *e*-organizations. However, we can define an *Electronic Institution* as a set of templates that can be adapted, parameterized or instantiated to build an *e*-organization.

It is composed by four levels of abstraction:

- the Abstract Level: where the statutes of the organization are defined in a high level of abstraction along with the first abstract norms.
- the Concrete Level: where abstract norms are iteratively concretized into more concrete norms, and the policies of the organization are also defined.
- the Rule Level: where the concrete norms are fully refined, linking the norms with the ways to ensure them. The policies defined in the previous level are refined.
- the Procedure Level: where all rules and policies are translated in a computationally efficient implementation easy to be used by agents.

The division of an *e*-institution into these four levels aims to ease the transition from the very abstract statutes, norms and regulations to the very concrete protocols and procedures implemented in the system, filling the gap among previous theoretical (abstract) approaches and practical (concrete) ones.

The abstract and concrete levels compose the Normative System of the *e*-institution. They define all the *obligations*, *rights* and *permissions* that should be applied in the system.

¹Part of the research described in this chapter has been published in [ANVS08, VSAN08]

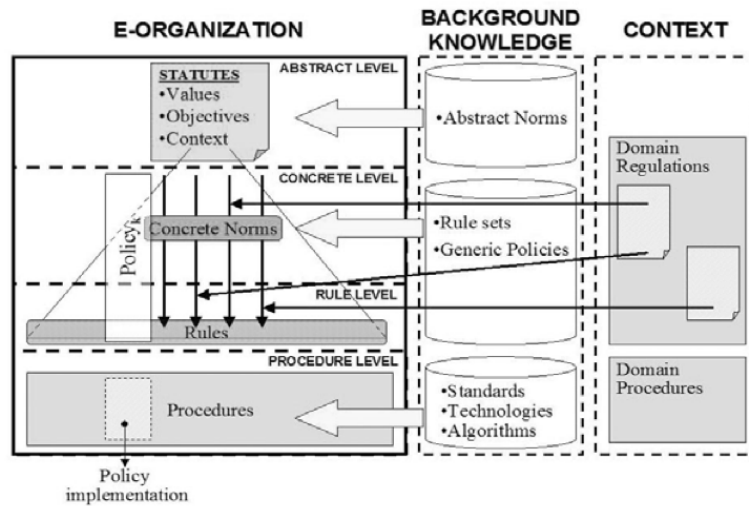


Figure 4.1: Multi-level architecture for Virtual Organizations.

The rule and procedure levels compose the Practical System of the *e*-institution, as they define how the agents inside the system should behave (the *praxis*) in order to follow the norms in the normative system, and how the *e*-institution can enforce the fulfillment of, at least, some of the norms.

The multi-level framework is depicted in figure 4.1. It depicts the refinement process where norms are defined, starting with the organization's statutes in the abstract level, and then progressively refined in more concrete norms in the concrete level. Then all those norms are translated into rules and, finally, implemented at the procedure level.

4.1.1 The Abstract Level: statutes, objectives, values and norms

The characteristic of institutions is that they enforce certain behaviors among members of the group of persons (or agents) that carry out their interactions within the jurisdiction (scope) of the organization. These behaviors must be of a type that contribute to a given pre-specified objective and set of values and norms. At the most abstract level these three elements can be found in the *statutes* of the organization. The statutes will indicate the main *objective* of the organization, the *values* that direct the fulfilling of this objective and they also point to the *context* where the organization will have to perform its activities.

At this level of abstraction, the highest, *values* fulfill the role of norms in the sense that they determine the actions that individuals should or should not take in a certain situation. Values are beliefs that individuals have about *what* is important, both to them and to society as a whole. A value, therefore, is a belief (right or wrong) about the way something should be. It should be apparent that values, by definition, always involve judgements (since they tell *how* something should be). In short, the values individuals hold are general behavioral guidelines. Values define beliefs about, for instance, what is acceptable or unacceptable, good or bad. However, values do not specify *how*, *when* or in

which conditions individuals should behave appropriately in any given social setup. This is the part played by abstract norms, concrete norms and rules.

The *objectives* of the organization can be represented as the goal of such organization. As far as the organization has control over the actions of the agents acting within that organization, it will try to ensure that they perform actions that will lead to the overall goal of the society.

The values of the organization can be described as desires. However, besides a formal syntax, this does not provide any meaning to the concept of *value*. In HARMONIA, the meaning of the values is defined by the abstract norms that contribute to this value. In an intuitive way we can see this translation process as follows:

$$\vdash_{org} V(\varphi) \longrightarrow O_{org}(\varphi)$$

meaning that, if an organization *org* has φ as one of its values, then such value can be translated in terms of an abstract norm (an obligation of the organization *org* to fulfill φ). But it usually happens that the value cannot be fully expressed by means of a single norm but several ones that contribute to a value. In HARMONIA a norm *contributes to a value* if fulfilling the norm always leads to states in which the value is more fully accomplished than the states where the norm is not fulfilled.

4.1.2 The Concrete Level: from abstract norms to concrete norms

In order to check norms and act against possible violations of the norms by the agents within an organization, the abstract norms have to be translated into actions and concepts that can be handled by the agents within such organization. Hence, *concrete norms* should refer to a) actions that are described in terms of the ontology of the organization and from which therefore the meaning and effect is known, or b) they pertain to situations that can be checked directly by the organization.

The abstract norms try to capture many different situations and therefore are *abstract* in several different ways:

1. They are referring to an abstract action that can be implemented in many ways
2. They use terms that are vague and that have to be defined separately
3. They abstract from temporal aspects
4. They abstract from agents and or roles
5. They refer to actions or situations that are not (directly) controllable and/or verifiable by the organization

4.1.3 The Rule Level: translating norms into rules

The translation from norms to rules marks the border between the *Normative System* in HARMONIA and the *Practical System*, from the normative dimension to the descriptive one. Such translation also implies a change in the language, from a deontic logic to a language more suitable to express actions and time constraints.

Meyer proposed in [Mey88] a reduction from deontic logic to a Propositional Dynamic Logic, that is based on the modal operator $[\]$. In his approach, deontic formulæ such as $O(\alpha)$, $F(\alpha)$ and $P(\alpha)$ are reduced to dynamic logic as follows:

$$F(\alpha) \mapsto [\alpha] V$$

This formula is the basis of the translation from norms to rules. Informally, it says that the action α is forbidden if and only if the performance of α leads necessarily to a state

in which the violation V holds. It can also be interpreted in the opposite direction: if the action α is forbidden, then the execution of α triggers the violation V .

Meyer, from this translation rule, also defines the translations for obligations and permissions:

$$P(\alpha) \equiv \neg F(\alpha) \mapsto \langle \alpha \rangle \neg V$$

$$O(\alpha) \equiv F(\neg\alpha) \mapsto [\neg\alpha] V$$

Informally, the first says that α is permitted if and only if α is not forbidden, so there is a way to perform α that leads to a state where violation V is not triggered. The second one expresses that α is obligatory if and only if not doing α is forbidden (and then it will trigger a violation).

In those cases where the norm expresses temporal relations among predicates, such relation can be translated to Dynamic Logic as follows:

$$O(\alpha < do(\beta)) \mapsto [\beta] done(\alpha)$$

$$O(\alpha < do(\beta)) \mapsto \neg done(\alpha) \rightarrow [\beta] V$$

The first reduction rule translates the temporal constraint of α being done before β starts with an expression in Dynamic Logic that states: “once action β is performed, it should always be the case that action α has been done”. The second reduction rule expresses the violation condition: “if action α has not been done, once action β is performed it always is the case that violation V occurs”. Therefore, concrete norms can be translated into two kinds of rules: *precedence rules* (rules defining precedence between actions) and *violation rules* (rules defining condition where a given norm is violated).

Once the translation from norms to rules has been done, some refinement can be done to those rules by adding more rules.

4.1.4 The Procedure level

The final step to build the e -institution is to implement the multi-agent system. There are two main approaches to be followed:

- The implementation of the rules in the rule level is done by using rule interpreters. This means that any agent entering the e -institution will need a suitable rule interpreter in order to behave properly. This will allow agents to read and reason directly about the rules.
- The implementation of the rules in the rule level is done by translating the rules into procedures to be easily followed by the agents. In this case the agents will be more efficient as they simply have to follow protocols described in a procedural way instead of reasoning about the rules. In the case of rules expressed in Dynamic Logic, this translation not only is easy to be done but also is principled by the following expression connecting Dynamic Logic formulæ with Hoare formulæ:

$$\{P\}\alpha\{Q\} \equiv P \rightarrow [\alpha]Q$$

In both cases, representing the norms by rules, procedures or any other description does not ensure that the agents will follow those descriptions. Also, the violations in the rule level should be translated in some detection mechanisms to check the behavior of the agents.

The alternative is to be able to accept both kinds of agents. That means that the *e*-institution provides, to all external agents entering into the organization, with the low-level protocols and also with the related rules. Those (standard) *Autonomous Agents* that are only able to follow protocols, will blindly follow them, while the ones that can also interpret the rules (that we call *Flexible Normative Agents*) will be able to choose among following the protocol or reasoning about the rules, or do both. With this approach the autonomy of the agents entering the *e*-institution is adapted to their reasoning capabilities.

The *Flexible Normative Agents*, proposed in [VS04], are a special kind of *Norm Autonomous Agents* where the agents can not only interpret and reason about rules and the related norms but also, in those cases where time efficiency is needed, follow a protocol.

In order to allow *Flexible Normative Agents* to switch from following low-level protocols to higher level rules and norms, there should be a link from procedures to rules, and from rules to norms. An advantage of the HARMONIA framework is that those links are created by the designer in the process from abstract to concrete norms to rules by means of the successive translations that are made. Those translations are links that allow to track, for instance, which abstract norms are related to a given rule. So the same should be made in the translation from rules to procedures.

4.1.5 Policies

In HARMONIA policies are elements that group the norms and rules at different levels together around specific topics. E.g. an *e*-institution trying to fulfill the role of the ONT will have to define, at least a Security policy for all the information about patients it manages. Policies go from the abstract level (where goals and values are defined) to the rule level (where those values are described in terms of rules).

4.1.6 Role hierarchy

The *objectives* in the organization's statutes can be represented in terms of goals (that we will call the *overall goal* of the society). As far as the organization has control over the actions of the agents acting within that organization, it will try to ensure that they perform actions that will lead to the overall goal of the society.

In HARMONIA, the overall goals are the ones that help to define the role hierarchy. The process starts assigning the overall goals to the *root* of the role hierarchy, then subgoals are identified and goals are distributed among the different actors of the system. Therefore, goal division depends on the context.

4.1.7 Ontologies

Apart from policies and roles, there is a third element that affects all four levels of the framework: ontologies. Ontologies are shared conceptualizations of terms and predicates in order to define a given domain. In our framework ontologies define the vocabulary to be used by all the agents in the *e*-institution. There are two main kinds of ontologies:

- *Domain ontologies*: these ontologies are focused to model the domain (e.g., organs and tissues) related with the *e*-institution activities.
- *Communication ontologies*: located in the procedure level, these ontologies define the predicates to allow agents to communicate one to the other (e.g., FIPA performatives).

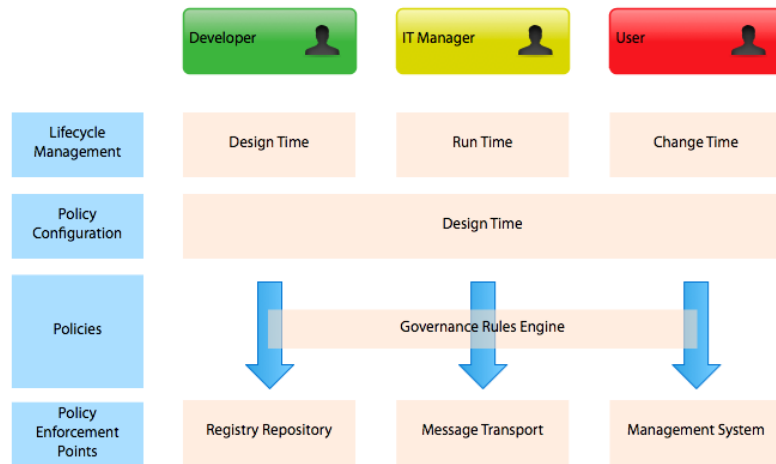


Figure 4.2: Key components of a SOA governance system

In HARMONIA *Domain ontologies* are defined in the abstract level and then extended in the following levels, as new norms and rules refer to new terms that are missing in the ontology.

4.2 ARCHITECTURE OF A SOA GOVERNANCE SYSTEM

As seen in Section 1.3.4, SOA Governance is a discipline that intends to provide methodologies for improving control and the application of policies in the full SOA lifecycle, from adoption to maintenance.

The company *webMethods* (a subsidiary of Software AG) has defined in a white paper [web06] a basic classification of the components needed for a SOA governance implantation, putting together methods found in the IT governance literature and the constraints and properties of a SOA. The key components identified for a SOA governance system are summarized in Figure 4.2 and described in the following sections.

4.2.1 SOA governance requirements

A SOA governance process, according to the interpretation of [web06], should follow an iterative sequential process, composed of four phases: architecture, design-time, run-time, and change-time. The following sections cover the requirements needed to fulfill each phase.

Architecture governance

The first requirement of SOA governance is architecture governance. Architecture governance is necessary to ensure that SOA as architecture evolves by design and not by accident. To the extent that it mirrors governance requirements in other areas of IT architecture, SOA architecture governance practices can be adapted from existing organizational architecture processes.

These include:

- Establishing organizational technology standards.
- Defining the high-level SOA architecture and topology, as well as the infrastructure capabilities that the SOA should incorporate.
- Determining the SOA platform strategy and making decisions about particular third-party products and technologies.
- Specifying the management, operations, and quality-of-service characteristics of the SOA, such as security, reliability, and availability.
- Establishing criteria for SOA project design reviews.

In addition, a key aspect of SOA architecture governance is defining a roadmap that will guide a smooth and thorough evolution of the architecture over time. The majority of organizational SOA strategies will involve overlaying and transforming the existing systems architecture in stages, rather than a whole replacement of the current infrastructure. Governance is needed to ensure that decisions made along the way align in a consistent direction and maintain the coherency of the SOA architecture.

Service lifecycle governance

A fundamental point in SOA [Erl04] is that it involves the creation of discrete, well-defined services that exist not only as building blocks of larger systems and applications, but more importantly as independent entities. As opposed to previous paradigms, SOA exposes standalone application functionality at a fine-grained level of granularity. A new form of governance is therefore needed: service-level lifecycle governance.

Service-level governance applies at the level of individual services and covers a wide range of requirements and situations. A useful approach to categorize the scope of activities associated with service-level governance is to consider the lifecycle of a service: beginning with its design, to its use in a run-time environment, to ongoing management and change of the service, as well as the people who have responsibilities in the governance of these services.

Design-Time Governance. Design-time governance is primarily an IT development function that involves the application of rules for governing the definition and creation of Web services. Policies might include ensuring that services are technically correct and valid, and that they conform to relevant organizational and industry standards. Examples of this type of validation might include checking that a service is compliant with the Web Services Interoperability (WS-I) profiles [NBCT06]: usage guidelines that ensure Web services implemented on different platforms are interoperable, by automatically verifying service schemas, validating namespaces, and other controls.

If an organization has a SOA governance infrastructure in place, in the form of software that facilitates the implementation of SOA governance practices, these checks can be invoked automatically when developers check services into a registry. In addition, approval and notification workflows can be triggered by a governance-enabled registry to ensure that services pass through pre-defined review and approval steps so that they meet architectural and organizational standards for business function encapsulation, reusability, reliability, and so on. By ensuring that these reviews are performed by appropriate members of the organization, it becomes possible to manage the quality and coherency of the service scenario effectively.

Key issues to consider include:

- Determining the fitness of a service, where fit is a function of the functionality that is encapsulated, the likelihood of reuse, and the importance of the service within the overall portfolio of services.
- Identifying which services to build against the previously existing organizational requirements.
- Ensuring the strategic design of services and validating that their interfaces and implementation conform to established design patterns and other organizational standards and practices.
- Establishing the governance standards to which different categories of services will be held, understanding that different levels of governance will be appropriate for different classes of services. Internal-use vs. services exposed to business partners, for example.

Other capabilities of design-time governance include fine-grained access control over assets in the registry, so that only authorized users are able to publish, search, and view services. In addition, the ability to label services and classify providers and consumers makes it possible to have some services visible to certain classes of service consumers and not others, a feature that is particularly important for partitioning access in a shared services model.

Run-Time Governance. Run-time governance is primarily of interest to IT operations. Governance at run-time revolves around the definition and enforcement of policies for controlling the deployment, utilization, and operation of deployed services. These run-time policies typically relate to non-functional requirements such as trust enablement, quality-of-service management, and compliance validation.

Examples of run-time governance include:

- Checking a service against a set of rules before it is deployed into production, for example, to ensure that only a certain message transport or specific schemas are used.
- Securing services so that they are accessible only to authorized consumers possessing the appropriate permissions, and that data is encrypted if required.
- Validating that services operate in compliance with prescribed organizational standards, in effect, to confirm that a service is not just designed to be compliant, but that its implementation is actually compliant.

A more specific case of run-time governance involves service-level monitoring and reporting. In order for the run-time SOA infrastructure to assess whether a given service is performing at the required level for a given consumer, in terms of response time, throughput, and availability, it is necessary to have an explicitly defined service level agreement between the service consumer and provider. SLAs can be expressed in terms of service contracts between consumer-provider pairs, and they establish the reference points for compliance monitoring and reporting by the SOA run-time environment. By tracking the actual performance of a service and comparing it to the requirements specified in the SLA, the system can identify non-compliant services and prompt remedial action. For example, automatically instantiating another instance of the service to improve load-balancing or alerting IT managers.

Change-Time Governance. Change is inevitable and, at some point, services deployed in the run-time environment will have to be changed to adapt to new requirements. Since the majority of services will be designed once and then modified several times over their lifetime, change-time governance, or in other words, the act of managing services through the cycle of change, is arguably more important in the long term than design-time governance.

Change-time governance requirements and considerations include:

- Understanding inter-service relationships and dependencies.
- Performing impact analysis to determine the implications of changing a particular service within the run-time environment.
- Managing the redeployment of services into the existing run-time environment.
- Managing service replacement through the design, coding, testing, and deployment stages.
- Managing changes to existing policies and service level agreements.

An important aspect of change-time governance is involvement of the organizational goals. This need arises from the fact that services exist to support organizational functions as well as the inter-organizational relationships and dependencies that are implicit in SOA, particularly when services are exposed and invoked across organizational boundaries. Since changes are generally initiated and driven by the requirements, users need to be participants in the governance lifecycle.

4.2.2 Technologies for SOA governance

If we follow the requirements structure presented in the previous Section 4.2.1, a SOA governance system should facilitate service-level governance across the lifecycle from design-time to run-time to change-time. It should allow policies to be defined and created, and provide mechanisms for these policies to be enforced at each phase of the service lifecycle. The main components of this system include:

- A *registry*, which acts as a central catalog of services.
- A *repository*, for storing policies and other metadata related to the governance of the services.
- *Policy enforcement points*, which are the agents that enact the actual policy enforcement and control at design-time, run-time, and change-time.
- A *rule engine* for managing the declaration of policies and rules and automating their enforcement.
- An *environment* for configuring and defining policies and for managing governance workflows across the service lifecycle.

Registry

A registry is usually identified as one of the first requirements of SOA adoption and registries play an important role in governance. In simple terms, a registry [Cle05] is a catalog or index that acts as the *system of record* for the services within a SOA. A registry is not designed to store the services themselves, but indicates their location by reference. Having a centralized catalog of services is significant from an organizational perspective because it enables the easy discovery, reuse, and management of services.

A SOA registry typically fulfills the following functions:

- Stores service descriptions, information about their endpoints and other technical details that a consumer requires in order to invoke the service, such as protocol bindings and message formats.
- Allows services to be categorized and organized.
- Allows users to publish new services into the registry and to browse and search for existing services.
- Maintains service history, allowing users to see when a service was published or changed.

As the place where services are made known within the SOA, a registry is also a natural management and governance point. For example, compliance requirements, such as conformance with the WS-I Basic Profile or the use of specific namespaces and schemas, might be imposed on services before they are allowed to be published in the registry. Or, as services are registered or changed, the registry also has the ability to trigger approval and change notification workflows so that people involved are alerted to changes. As such, a robust registry is an important component of any SOA governance solution.

Another important factor is the interoperability of the registry with other components of the SOA infrastructure. OASIS provides a platform-independent standard for registry interoperability known as UDDI (Universal Description, Discovery, and Integration). UDDI [Bel01] defines a Web services-based programming interface that allows different consumer applications, tools, and run-time systems to query the registry, discover services, and interact as required to provide management and governance capabilities. While it is not a pre-requisite for a SOA registry to be based on UDDI, it is the most commonly adopted standard and ensures the greatest degree of compatibility with other products in the environment.

Repository

While the registry plays a central role in policy enforcement, the registry itself does not provide sufficient context for the whole set of SOA governance requirements. For example, policies, in the form of rules and restrictions that are enforced on services, and consumer/provider service level agreements are generally not constructs that are stored in a registry. Thus another data store, usually referred to as a repository, is needed for storing governance-related artifacts and supporting the full complexity of managing service metadata throughout the service life cycle. The term *repository* is used in many different contexts, but in the context of SOA governance, the repository can be thought of as a centrally-managed policy store.

Among other things, a governance repository should support the following capabilities:

- An ontology for representing and storing organizational and regulatory policies that can be translated into rules that are enforced by the SOA governance system. It should be possible for policies and rules to be interpreted by people or machines, and sometimes both, as appropriate.
- Audit capabilities for tracking the trail of changes and authorizations applied to assets within the repository context.
- Identity management capabilities and role-based access controls to ensure that only appropriate parties have access to policies.
- A notification system and content validation capabilities to provide additional assurances that policies are well-formed, consistent, and properly applied.

The requirement for a logically centralized repository is particularly important for codifying and enforcing a single *official* set of policies across the organization. However, the actual repository itself may have a federated architecture for scalability and to enable the use of the repository across different geographic regions, multiple lifecycle instantiations, and cross-organizational boundaries.

Policy Enforcement Points

The places where policies are actually applied and enforced, the policy enforcement points, change depending on the lifecycle stage. During design-time, the registry and the repository form together the main point of enforcement². During run-time, policies are generally enforced by the underlying message transport system that connects service providers with consumers. Finally, during change-time, policies are typically enforced by the IT management system.

Design-Time Enforcement: Registry and Repository. Since the registry/repository is the system of record for both service interfaces as well as attributes and metadata associated with them, it provides a logical point at which to enforce policies about the design of these particular artifacts. Design-time policies are typically applied as artifacts, which could include WSDL files, schema definitions, process models, and project documentation. They are checked into the registry/repository.

The following features are desirable in the design-time policy enforcement point:

- *Identity management:* in order to establish rights and responsibility in the registry/repository it is first necessary to identify users, service consumers, and other participants. Identity is also important for metering usage, logging for audit purposes, and applying approval requirements and other governance processes on an individual or role basis.
- *Access control:* coupled with identity management, the system should offer fine-grained access configurations over all aspects of registry/repository assets. This includes the ability to secure policies, governance processes, and classifications.
- *Automated notifications and approvals:* the ability to trigger events in response to management activities in the registry/repository allows alerts, approval processes, content validation scans, and other actions to be automated. These triggers might be applied either before or after the interaction in question. For example, a policy might be established that a design review approval is needed before an object is created in the registry/repository.
- *Content validation:* content should be scanned and validated according to their type and pre-configured compliance checks. Common validations include WSDL validation, XML schema validation, testing for namespace violations, schema validation, and other interoperability-related scans. For example, service consumers expect interfaces to be well-formed and interoperable, so the registry/repository should automate the process of scanning and assuring that WSDL documents are well-formed and conformable with WS-I interoperability profiles.

²The white paper strongly suggests that the registry and the repository are to be implemented in the same software unit, as they should maintain a consistent view of service definitions, service versions, consumer and user identities, and other information. See [web06].

- *Audit trails*: a fundamental capability for establishing accountability is the ability to track interactions between participants and the registry/repository, along with the sequence and details of those activities. This record can be used for governance enforcement *after the fact* and to establish usage patterns for guiding process improvements.

Run-Time Enforcement: Message Transport. Run-time policy enforcement relies on a SOA infrastructure that is able to exercise policy enforcement in a way that is transparent to, and independent of, the service providers and consumers. This is achieved through an agent or intermediary that resides between provider and consumer and a registry/repository that addresses both the needs of run-time service discovery as well as policy enforcement.

The intermediary interacts with the registry/repository to find services and their run-time policies and enforces the policies during the execution of the service. In a SOA, the run-time system is typically a message transport or mediation layer [SHLP05]. The message transport brokers transactions between service provider and service consumer and frequently offers additional functions such as data transformation, message queuing, reliable messaging, and other operational capabilities.

Without SOA, the ability to control and manage applications in this manner is restricted both by the scope and the capabilities of the underlying platform. When different applications are integrated, it is generally infeasible to apply a common policy context to the integrated result. A typical challenge is enforcing access security when two applications with different user communities are integrated. With the intermediation provided by the message transport, it becomes possible for a distributed network of services to share a common policy-managed context.

Since run-time policies are typically applied to messages that flow across the message transport system, the types and level of sophistication of run-time policies that can be defined and enforced depend on the capabilities of the underlying intermediary. Desirable areas of policy configuration include the following:

- *Consumer identification and security*: identifying consumer applications to prevent unauthorized access to services; configuring the security of services at run-time and enforcing policies such as encryption, digital signatures, and logging for tracing and tracking.
- *Routing rules*: configuring run-time routing rules to address performance, version management, and other operational requirements. Variations include content-based routing, version-based routing, and preferential quality-of-service routing.
- *Transformation rules*: translating between different message transports and technology protocols to facilitate application connectivity, or transforming data between consumer and provider.
- *Service Level Agreement (SLA) management*: policies for managing performance and availability to match the requirements of an SLA, for example, routing a request to a backup service in the event of a failure of the primary service provider, or balancing the request load across additional back-end service to improve performance.
- *Logging, monitoring, and alerting*: collecting service-level data and establishing rules based on aggregate counters for response time, throughput, errors, and other transaction data so that alerts can be generated when there are violations to predefined SLAs.

Finally, while the intermediary and registry/repository are logically decoupled, a dependency exists to the extent that the intermediary has to understand and interpret the policies defined in the registry/repository. As such, it is advantageous to have a message transport system and registry/repository that are interoperable out of the box. Otherwise, this is an integration issue that the implementer has to address.

Change-Time Enforcement: IT Management System. Change-time enforcement relies to a greater extent on IT change management practices and procedures than on enforced control points. Unlike previous software paradigms where an application package enters a support or maintenance phase once put into production, SOA involves a dynamic network of interdependent services that are in an ongoing state of adaptation and optimization. Since services, transactions, and SOA events of interest can be monitored by the IT management system, it is a logical source of run-time information that can be fed back into the registry/repository to facilitate the orderly evolution of the SOA environment.

This information might include:

- SLA-related metrics, such as the average response time, availability, or throughput of a specific service.
- Process-related metrics in the form of Key Performance Indicators (KPIs), which associate services with user-defined business process metrics (e.g., average order amount).
- Activity monitoring, alerting, and notification events related to business-level exceptions.

Information such as this can be used to optimize service delivery during the change-time cycle by guiding adjustments in policies, service levels, or in the services themselves. Changes to services will require the change-time governance practices described earlier to be put into effect, for example, performing an impact analysis to assess the implications of changing a service and dealing with the resulting version management issues.

As with integration between the message transport and the registry/repository, it is beneficial to have out-the-box linkages between the registry/repository and the management system so that data flows seamlessly between the two without the need for additional integration.

Governance rules engine

A rules engine is not strictly a requirement of a SOA governance system, but incorporating rules engine technology within the registry/repository enables a significant degree of flexibility and automation, while reducing the reliance on humans to perform mechanical governance tasks.

Rules are typically associated with events, while the rules engine handles the firing and chaining of rules. The rules engine could automate the process of setting and resetting access control switches at lifecycle milestones such as when a service is promoted from development into testing or production. A rules engine also provides the basis for creating complex policies based on reusable templates.

In addition to automating governance tasks, the rules engine can also help to deal with policy federation [MTWM07], or the ability to allow multiple policy authors and authorities. This is an important use case for SOA adoption where governance policies might not be authored and controlled by a single department or organization. A more robust model,

which is the basis for policy federation, is to enable both centralized as well as distributed policy creation. Policy federation requires the establishment of guidelines and rules for reconciling policies that come into conflict, and the rules engine assists in the execution of these rules.

Lifecycle Management

The final key ingredient of a SOA governance system is the user environment that presents the human interface to the registry/repository and which incorporates the governance lifecycle processes and workflows. Typically, the process workflow includes the following steps:

- Publishing of a service by an authorized provider.
- Discovery of a service by a potential consumer.
- Requesting use of the service by the consumer.
- Agreeing on the terms of delivery of the service.
- Authorizing the consumer.
- Provisioning of the service.
- Monitoring of the service delivery.

Related to each of these steps, organizations might define approval and notification workflows, exception alerts, and a variety of other process steps.

4.3 A CONTRACTUAL INSTITUTION- AND PROVENANCE-BASED NORM ENFORCEMENT MECHANISM

In this section we describe our proposal 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.

Apart from what we saw in Section 1.3.4, academic research on SOA Governance is still not abundant. However, there are already some interesting proposals of models [DW06], methodologies [ZLK⁺07, PvdH06] and frameworks [KMLS07]. Here we present a novel approach to the topic based on flexible normative enforcement via landmark monitoring.

4.3.1 A Normative framework based on Norms and Landmarks

As explained in Section 4.1, we use HARMONIA as the basis for our adapted normative framework for SOA, 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 [AGVSD05].

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 behavior (that is, agents are seen as *black boxes*). The enforcement

<i>Norm</i>	<i>OTM : N37</i>
<i>Condition</i>	OBLIGED(<i>hospital</i> DO <i>ensure_compatibility(organ, recipient)</i> BEFORE(<i>allocatorDOassign(organ, recipient)</i>))
<i>Violation condition</i>	NOT(<i>done(ensure_compatibility(organ, recipient)</i> AND <i>done(assign(organ, recipient)</i>)
<i>Sanction</i>	<i>inform(board, "NOT(done(ensure_compatibility(organ, recipient))ANDdone(assign(organ, recipient))")</i>
<i>Repairs</i>	{ <i>stop_assignment(organ)</i> ; <i>assert</i> (NOT(<i>done(ensure_compatibility(organ, recipient)</i> BEFORE <i>done(assign(organ, recipient)), p_store</i>); <i>wait(asserted</i> (<i>ensure_compatibility(organ, recipient)</i>)); <i>resume_assignment(organ)</i> ;} }

Figure 4.3: Example of an OTMA norm

of the norms in an *e*-Institution is achieved through a special kind of agents, the *Enforcement Agents*, which monitor the behavior of the agents, detect violations and check the compliance of the sanctions.

Norm language

We use a language for substantive norms [ADGC⁺06] which is an evolution of the original norm language in HARMONIA, and which will be, in practice, adapted from the clause representation for the language presented in Section 3.3. Its central element is the norm condition, based on 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 4.3. 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 behavior (i.e., avoid the system to enter in a undesirable, illegal state because of a failure in one of the agents).

<i>Norm</i>	<i>OTM : N37</i>
<i>Violation condition</i>	NOT(<i>done(ensure_compatibility(organ, recipient))</i>)AND <i>done(assign(organ, recipient))</i>)
<i>Detection condition</i>	(NOT(<i>asserted(ensure_compatibility(organ, recipient), t1)</i> AND <i>asserted(assign(organ, recipient), t2)</i>)OR (<i>asserted(ensure_compatibility(organ, recipient), t1)</i> AND <i>asserted(assign(organ, recipient), t2)</i> AND(< <i>t2 t1</i>)))

Figure 4.4: Example of a violation handling rule

Control Landmarks

Landmarks [Ald07] 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 propose to 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 4.3 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 4.4 shows an example of how these p-assertions can be then used to detect a violation of the norm in Figure 4.3.

4.3.2 An architecture proposal for norm enforcement in SOA

In this section we introduce our proposal for a generic Provenance-based norm enforcement architecture. Although the 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 platform³. The global picture of this architecture is shown in Figure 4.5. When application agents

³This issue is already planned to be done in the PhD proposed here.

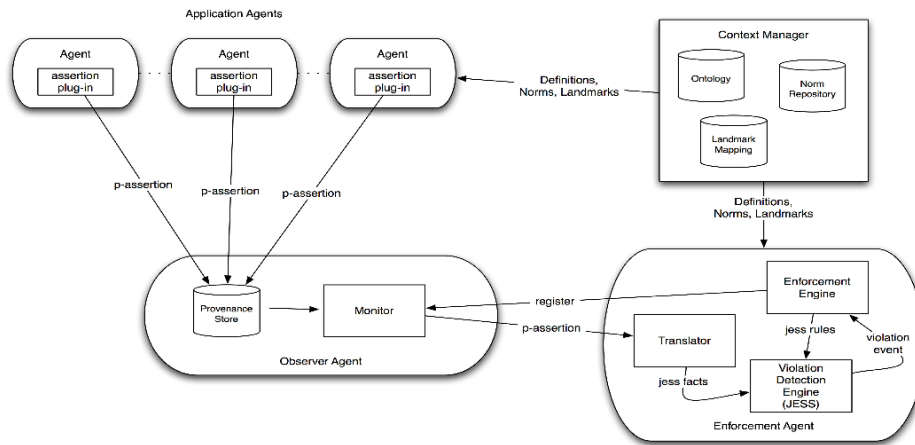


Figure 4.5: A generic Provenance-based norm enforcement architecture

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, HARMONIA, 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.

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 interoperability 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' behavior. 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. 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 agent's 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 4.6: An example of translation rule from p-assertion to Jess *asserted* fact

The Translator obtains the translation rules from the Context Manager. In Figure 4.6 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 *opencontent* 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 4.7. The rule shown in the figure takes as input two facts: the fact generated (using the translation rule shown in Figure 4.6) 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.

```
(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 4.7: An example of violation detection rule in Jess

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 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 4.3, all the measures should be executed by the Enforcement Agents, as they are all institutional.

4.3.3 Related work to our proposal

AMELI [Est03] 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 [ADGC⁺06, Ald07] 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 [CVP07] only considers messages as observable events.

[APL⁺06] 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 behavior 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 [GJM⁺06] 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.

4.4 CONCLUSIONS

This Chapter has introduced our proposal for a PhD thesis. The objectives of this proposal are, as seen in Section 1.5:

- To design and implement an open source norm enforcement framework using the latest technology based on SOA available.
- To apply and generalize, on the design of this framework, the state of the art in Electronic Institutions.
- To integrate this framework in the SOA governance methodology lifecycle.
- To create a mapping between the operational representation of norms and both orchestration and choreography languages.
- To benchmark the resulting framework against other SOA governance options available in the industry.
- To find and integrate the ideas and concepts shared by both Governance and Institutional theoretical frameworks but developed independently.
- To adapt the existing theories about Electronic Institutions to the wider scope of SOA.
- To contribute the advances done in regard to the SOA generalization to the theory of Electronic Institutions.

In the previous Chapters we have introduced our previous steps of work towards these objectives. More concretely, the notable achievements have been:

- To integrate a monitoring mechanism for SOA, based on a provenance architecture, into a distributed complex use case on top of agents and services (Chapter 2).

- To create a language for electronic contracts, which include a high level formalism for representing norms as clauses (Section 3.3).
- To design and implement a middleware for the management of these electronic contracts, and therefore giving services the capability to create electronic contractual institutions (Section 3.4).
- To identify common features of electronic institutions and SOA governance (Sections 4.1 and 4.2).

Taking all this into account, we have proposed an architecture for norm enforcement that covers the needs of SOA governance and manages electronic institutions in SOA (Section 4.3). From the work already done and presented there are still some further steps to undertake in order to achieve the objectives of this PhD proposal.

First of all, the proposal is solidly integrated with the concept of provenance. However, the aspects concerning the use of electronic contracts and the management of electronic contractual institutions has not been incorporated to the architecture proposed yet. For example, the enforcement mechanism is designed as generic enough as to be built on top of the CONTRACT middleware, but this integration has to be done. This is a main issue to be included in the working plan.

State of the art in service-oriented cooperation already defines frameworks for orchestration and choreography. Rather than defining a new one, we should try to integrate them to our architecture, by defining an architectural mapping between the operational representation of norms and this low-level representation of service processes and workflows.

Therefore, the architecture is still to be refined and fully formalized. The next step after that will be to build a preliminary version of this architecture and adapt our use case on top of it. This implementation should be deployed in a distributed SOA-based system, as heterogeneous as possible, and tested against the previous versions of our use case.

This testing will make us define some metrics about how to evaluate the performance of the system. These metrics should involve institutional parameters, so Institutional Theory will be analyzed for this matter.

As our architecture is intended to be useful in SOA governance, we will incorporate the implementation into governance lifecycles defined in the SOA governance methodology and tested against other alternatives.

Finally, Institutional Theory and the literature on Governance share many concepts that have been treated independently. If the integration between electronic institutions and SOA governance proves to be successful, it should be interesting to try to unify these concepts found on both frameworks.

Working Plan

The work to be done is summarized in this chapter. It has been divided in three main steps: the refinement of the architecture, its implementation, and finally its deployment.

1. Refinement of the architecture

- a) Map SOA Governance and Institutional Theory common concepts, in order to generalize those concepts in our architecture as possible.
- b) Generalization and adaptation of the contracting language to the HARMONIA framework.
- c) Integration of the contracting framework to the architecture.
- d) Identify issues and drawbacks of the architecture and solve them.
- e) Formalization of the architecture as an electronic institutional framework, comparing it to other alternatives.
- f) Mapping between SOA orchestration and choreography frameworks and the operational representation of norms of our architecture.
- g) Extract and define metrics, from Institutional Theory, for the evaluation of the architecture.

2. Implementation

- a) Research on the suitable technologies for the implementation.
- b) Adaptation of the contracting middleware for its use in the architecture.
- c) Implementation of the components of the architecture.
- d) Integration of the use case in the implementation.

3. Deployment

- a) Design of the test cases.
- b) Creation of testbeds for the deployment, which should reflect as many cases found in SOA systems as possible, but being as generic as possible as well.
- c) Deployment of the use case in the testbeds.
- d) Benchmark of the results of the testbed executions.

Jan-Apr 09	May-Aug 10	Sep-Dec 10	Jan-Apr 10	May-Aug 10
1a	1f	2c	3a	3f
1b	1g	2d	3b	3g
1c	2a		3c	
1d	2b		3d	
1e				

Table 5.1: Time planning.

- e) Integration of the architecture in a SOA governance methodology lifecycle.
- f) Benchmark of the results of the architecture used as a SOA governance mechanism.
- g) Once the results are analyzed, find issues concerning the performance of the architecture.

The time planning for these tasks is summarized in Table 5.1.

Bibliography

- [ACD⁺05] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu, *Web Services Agreement Specification (WS-Agreement) Version 2005/09*, Global Grid Forum (2005).
- [ADGC⁺06] Huib Aldewereld, Frank Dignum, Andrés García-Camino, Pablo Noriega, Juan Antonio Rodríguez-Aguilar, and Carles Sierra, *Operationalisation of norms for usage in electronic institutions*, Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (2006).
- [AGVSD05] Huib Aldewereld, Davide Grossi, Javier Vázquez-Salceda, and Frank Dignum, *Designing Normative Behaviour by the Use of Landmarks*, Proceedings of AAMAS-05 International Workshop on Agents, Norms and Institutions for Regulated Multiagent Systems (2005).
- [Ald07] Huib Aldewereld, *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*, PhD Thesis (2007).
- [ANOB⁺07] Sergio Alvarez-Napagao, Nir Oren, Jiri Biba, Sofia Panagiotidi, Roberto Confalonieri, Monika Solanki, Martin Dehn, Javier Vázquez-Salceda, Martin Kollingbaum, Steven Willmott, and Michal Jakob, *Contract based Electronic Business Systems State of the Art*, IST CONTRACT Project Deliverable (2007), 257.
- [ANVS08] Sergio Alvarez-Napagao and Javier Vázquez-Salceda, *Using Provenance to implement a Norm Enforcement Mechanism for Agent-Mediated Healthcare Systems*, Proceedings of the Fifth Workshop on Agents Applied in Health Care at AAMAS'08, Estoril, Portugal (2008), 8.
- [ANVSK⁺06] Sergio Alvarez-Napagao, Javier Vázquez-Salceda, Tamás Kifor, László Z Varga, and Steven Willmott, *Applying provenance in distributed organ transplant management*, International Provenance and Annotation workshop (IPAW 2006), 3-5 May 2006, Chicago, USA, ISBN 978-3-540-46302-3 (2006).
- [APL⁺06] Ronald Ashri, Terry Payne, Michael Luck, Mike Surridge, Carles Sierra, Juan Antonio Rodríguez-Aguilar, and Pablo Noriega, *Using Electronic Institutions to secure Grid environments*.
- [Ari57] Aristotle, *Ethics*.

- [Arn99] K Arnold, *The Jini architecture: dynamic services in a flexible network*, 36th Annual Conference on Design Automation (DAC'99) (1999), 157–162.
- [Bal90] Wolfgang Balzer, *A Basic Model For Social Institutions*, The Journal of Mathematical Sociology 16(1) (1990), 1–29.
- [BBB⁺02] Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma, and Scott Williams, *Web Services Conversation Language (WSCL) 1.0*, W3C Note (2002).
- [BD06] R Barga and L Digiampietri, *Automatic Generation of Workflow Provenance*, LECTURE NOTES IN COMPUTER SCIENCE (2006).
- [Bel01] Thomas A Bellwood, *UDDI-A Foundation for Web Services*, Proceedings of XML Conference & Exposition. Orlando (2001).
- [BGH⁺06] Uri Braun, Simson Garfinkel, David A Holland, Kiran-Kumar Muniswamy-Reddy, and Margo I Seltzer, *Issues in Automatic Provenance Collection*, LECTURE NOTES IN COMPUTER SCIENCE (2006).
- [BHM⁺04] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Christopher Ferris, and David Orchard, *Web Services Architecture*.
- [BKT01] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan, *Why and Where: A Characterization of Data Provenance*, International Conference on Database Theory (2001).
- [Bol06] Harold Boley, *The RuleML Family of Web Rule Languages*, LECTURE NOTES IN COMPUTER SCIENCE (2006).
- [Bou08] Kostas Bougiouklis, *PANDA: Collaborative Process Automation Support using Service Level Agreements and Intelligent dynamic Agents in SME clusters*, 6.
- [BT01] Wolfgang Balzer and Raimo Tuomela, *Social Institutions, Norms, and Practices*, Social Order in Multiagent Systems (2001).
- [CANP⁺08] Roberto Confalonieri, Sergio Alvarez-Napagao, Sofia Panagiotidi, Javier Vázquez-Salceda, and Steven Willmott, *A Middleware Architecture for Building Contract-Aware Agent-Based Services*, Proceedings of the International Workshop on Service-Oriented Computing: Agents, Semantics, and Engineering -SOCASE'08-, at AAMAS'08, Estoril, Portugal, ISBN 978-3-540-79967-2 (2008).
- [CC01] Rosaria Conte and Cristiano Castelfranchi, *Are Incentives Good Enough To Achieve (Info) Social Order?*, Social Order in Multiagent Systems, Springer, ISBN 0792374509 (2001).
- [CDSL03] John L Colley, Jacqueline L Doyle, Wallace Stettinius, and George Logan, *Corporate Governance*, McGraw-Hill Professional (2003).
- [CGD⁺99] M Condry, U Gall, P Delisle, S Microsyst, and P Alto, *Open Service Gateway architecture overview*, Industrial Electronics Society (1999).
- [Cle05] Luc Clement, *Risks of Running SOA without Registry*, Loosely Coupled (2005).

- [CO00] Henrique Lopes Cardoso and Eugenio Oliveira, *Using and Evaluating Adaptive Agents for Electronic Commerce Negotiation*, Proceedings of the International Joint Conference (2000).
- [CRA00] Ulises Cortés and Juan Antonio Rodríguez-Aguilar, *Trading agents in auction-based tournaments*, Special issue on Intelligent Agents, INFORMATIQUE 1/2000:39-50 (2000).
- [CVP07] Owen Cliffe, Marina De Vos, and Julian Padget, *Embedding Landmarks and Scenes in a Computational Model of Institutions*, COIN 2007@DURHAM (2007).
- [CWW00] Yingwei Cui, Jennifer Widom, and Janet L Wiener, *Tracing the lineage of view data in a warehousing environment*, ACM Transactions on Database Systems (TODS) (2000).
- [Del00] Chrysanthos Dellarocas, *Contractual Agent Societies: Negotiated shared context and social control in open multi-agent systems*, Social Order in Multiagent Systems, Springer, ISBN 0792374509 (2000).
- [Die05] Jens Dietrich, *The Mandarax Manual*, Available at: <http://fisheye.cenqua.com> (2005).
- [dL04] Mark d’Inverno and Michael Luck, *Understanding Agent Systems*, Springer (2004).
- [DMWD02] Virginia Dignum, John-Jules Ch Meyer, Hans Weigand, and Frank Dignum, *An Organization-oriented Model for Agent Societies*, Proceedings of International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA’02), at AAMAS, Bologna, Italy (2002), 20.
- [DW06] Patricia Derler and Rainer Weinreich, *Models and Tools for SOA Governance*, TEAA 2006 (2006).
- [DWX02] Virginia Dignum, Hans Weigand, and L Xu, *Agent Societies: Towards Frameworks-Based Design*, LECTURE NOTES IN COMPUTER SCIENCE (2002).
- [EPS02] Marc Esteva, Julian Padget, and Carles Sierra, *Formalizing a language for institutions and norms*, Intelligent Agents VIII (2002).
- [Erl04] Thomas Erl, *Service-oriented architecture*, Prentice Hall (2004).
- [Est03] Marc Esteva, *Electronic Institutions: from specification to development*, PhD Thesis (2003).
- [FFMM94] Tim Finin, R Fritzson, D McKay, and R McEntire, *KQML as an agent communication language*, Proceedings of the Third International Conference on Information and Knowledge Management (CIKM’94), ACM Press (1994).
- [FHD08] Larry Fulton, Randy Heffner, and David D’Silva, *The Forrester Wave: SOA Service Life-Cycle Management, Q1 2008*, Forrester Research (2008).
- [Fin95] Lawrence S Finkelstein, *What Is Global Governance*, Global Governance (1995).
- [fIPA00] Foundation for Intelligent Physical Agents, *FIPA SL Content Language Specification*, Foundation for Intelligent Physical Agents (2000).

- [fIPA02] Foundation for Intelligent Physical Agents, *FIPA ACL Message Structure Specification*, Foundation for Intelligent Physical Agents (2002).
- [FJK04] Ian Foster, Nicholas R Jennings, and Carl Kesselman, *Brain meets brawn: why grid and agents need each other*, 3rd International Conference on Autonomous Agents and Multi-Agent Systems (2004).
- [Fut06] Joe Futrelle, *Harvesting RDF Triples*, Provenance and Annotation of Data, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, ISBN 978-3-540-46302-3 4145/2006 (2006), 64–72.
- [FVWZ02] Ian Foster, J Voekler, Michael Wilde, and Yong Zhao, *Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation*, Proceedings of the 14th Conference on Scientific and Statistical Database Management (2002).
- [FVWZ03] Ian Foster, Jens-S Vöckler, Michael Wilde, and Yong Zhao, *The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration*, Conference on Innovative Data Systems Research (2003).
- [GGGRA99] Pere Garcia, Eduard Giménez, Lluís Godó, and Juan Antonio Rodríguez-Aguilar, *Bidding Strategies for Trading Agents in Auction-Based Tournaments*, AMET-98, Lecture Notes in Computer Science, LNAI 1571 (1999), 151–165.
- [GGRS02a] Jonathan Gelati, Guido Governatori, Antonino Rotolo, and Giovanni Sartor, *Declarative power, representation, and mandate. a formal analysis*, Procs. of JURIX (2002).
- [GGRS02b] Guido Governatori, Jonathan Gelati, Antonino Rotolo, and Giovanni Sartor, *Actions, institutions, powers. Preliminary notes*, International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02) (2002).
- [GJM⁺06] Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau, *An Architecture for Provenance Systems*, PROVENANCE consortium (2006).
- [GLM04] Paul Groth, Michael Luck, and Luc Moreau, *Formalising a protocol for recording provenance in grids*, Proc. of the UK OST e-Science second All Hands Meeting (2004).
- [IHJ⁺07] Mamdouh Ibrhaim, Kerrie Holley, Nicolai M Josuttis, Brenda Michelson, Dave Thomas, and John deVadoss, *The future of SOA: what worked, what didn't, and where is it going from here?*, OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion (2007).
- [KF04] Carl Kesselman and Ian Foster, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann (2004).
- [KL03] Alexander Keller and Heiko Ludwig, *The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services*, Journal of Network and Systems Management (2003).
- [KMLS07] Mira Kajko-Mattsson, Grace Lewis, and Dennis B Smith, *A Framework for Roles for Development, Evolution and Maintenance of SOA-Based Systems*, SD-SOA '07 (2007).

- [KP08] L Frank Kenney and Daryl C Plummer, *Magic Quadrant for Integrated SOA Governance Technology Sets, 2007*, Gartner RAS Core Research Note G00153858 (2008), 16.
- [KS74] Stig Kanger and Sören Stenlund, *Logical Theory and Semantic Analysis: Essays Dedicated to Stig Kanger on His Fiftieth Birthday*, Springer, ISBN 9027704384 (1974).
- [KS04] Alexander Kozlenkov and Michael Schroeder, *PROVA: Rule-Based Java-Scripting for a Bioinformatics Semantic Web*, Data Integration in the Life Sciences, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, ISBN 978-3-540-21300-0 2994/2004 (2004), 17–30.
- [KVAN⁺06] Tamás Kifor, László Z Varga, Sergio Alvarez-Napagao, Javier Vázquez-Salceda, and Steven Willmott, *Privacy Issues of Provenance in Electronic Healthcare Record Systems*, First International Workshop on Privacy and Security in Agent-based Collaborative Environments (PSACE2006), Hakodate, Japan (2006).
- [KVVS⁺06] Tamás Kifor, László Z Varga, Javier Vázquez-Salceda, Sergio Alvarez-Napagao, Steven Willmott, Simon Miles, and Luc Moreau, *Provenance in Agent-mediated Healthcare Systems*, IEEE Intelligent Systems, ISSN 1541-1672 21 (2006), no. 6, 38–46.
- [LDK04] Heiko Ludwig, Asit Dan, and Robert Kearney, *Cremona: an architecture and library for creation and monitoring of WS-agreements*, Proceedings of the 2nd international conference on Service oriented computing, New York, NY, USA, ISBN:1-58113-871-7 (2004), 65–74.
- [Lin94] Lars Lindahl, *Stig Kanger's Theory of Rights*, Collected Papers of Stig Kanger with Essays on His Life and Work, Springer, ISBN: 1402001118 (1994).
- [LKD⁺03] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P King, and Richard Franck, *Web Service Level Agreement (WSLA) Language Specification*, IBM Corporation (2003).
- [Lok99] Gert-Jan C Lokhorst, *Ernst Mally's Deontik (1926)*, Notre Dame Journal of Formal Logic 40 (1999), no. 2, 273–282.
- [Lyn96] Nancy A Lynch, *Distributed Algorithms*, Morgan Kaufmann (1996).
- [MACM01] Amélie Marian, Serge Abiteboul, Grégory Cobéna, and Laurent Mignet, *Change-Centric Management of Versions in an XML Warehouse*, Proceedings of VLDB 2001 (2001).
- [Mey88] John-Jules Ch Meyer, *A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic*, Notre Dame Journal of Formal Logic 29 (1988), no. 1, 109–136.
- [MI06] Luc Moreau and John Ibbotson, *The EU Provenance Project: Enabling and Supporting Provenance in Grids for Complex Problems (Final Report)*, PROVENANCE consortium (2006).
- [Mil99] Robin Milner, *Communicating and Mobile Systems: The Pi Calculus*, Cambridge University Press (1999).

- [MKR⁺07] L Matyska, A Krenek, M Ruda, J Sitera, and D Kouril, *Job Tracking on a Grid—the Logging and Bookkeeping and Job Provenance Services*, CESNET technical report number 9/2007 (2007).
- [MLM⁺06] C Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, and Rebekah Metz, *Reference Model for Service Oriented Architecture 1.0*, OASIS Committee Specification 1 (2006), 31.
- [MM04] Robert A G Monks and Nell Minow, *Corporate Governance*, Blackwell Publishing (2004).
- [MP99] Paul McNamara and Henry Prakken, *Norms, Logics and Information Systems: New Studies in Deontic Logic and Computer Science*, IOS Press (1999).
- [MRHBS06] Kiran-Kumar Muniswamy-Reddy, David A Holland, Uri Braun, and Margo I Seltzer, *Provenance-Aware Storage Systems*, Annual Tech '06: 2006 USENIX Annual Technical Conference, USENIX Association (2006), 43–56.
- [MS99] Noyda Matos and Carles Sierra, *Evolutionary Computing and Negotiating Agents*, AMET-98, LNAI 1571, Lectures Notes on Computer Science, Springer-Verlag Berlin Heidelberg (1999), 126–150.
- [MT95] Yoram Moses and Moshe Tennenholtz, *Artificial Social Systems*, Computers and Artificial Intelligence (1995).
- [MTWM07] Michael Menzel, Ivonne Thomas, Christian Wolter, and Christoph Meinel, *SOA Security-Secure Cross-Organizational Service Composition*, Proceedings of the Stuttgarter Softwaretechnik Forum (SSF), Fraunhofer IRB-Verlag, Stuttgart, Germany, ISBN: 978-3-8167-7493-8 (2007), 41–53.
- [MW91] John-Jules Ch Meyer and R Wieringa, *Deontic Logic in Computer Science: Normative System Specification*, Procs. of MAAMAW'01 (1991).
- [NBCT06] Hamid R Motahari Nezhad, Boualem Benatallah, Fabio Casati, and Farouk Toumani, *Web Services Interoperability Specifications*, Computer, IEEE Computer Society, Los Alamitos, CA, USA, ISSN: 0018-9162 39 (2006), no. 5, 24–32.
- [NNS90] Douglass C North, Asbjørn Sonne Nørgaard, and Richard Swedberg, *Institutions, Institutional Change and Economic Performance*, Cambridge University Press (1990).
- [Nor97] Pablo Noriega, *Agent-Mediated Auctions: The Fishmarket Metaphor*, IIIA Phd Monography (1997), no. 8.
- [OML⁺08] Nir Oren, Simon Miles, Michael Luck, Sanjay Modgil, Nora Faci, Sergio Alvarez-Napagao, Javier Vázquez-Salceda, and Martin Kollingbaum, *Contract based Electronic Business Systems Theoretical Framework*, IST CONTRACT Project Deliverable (2008), 126.
- [OPVSM08] Nir Oren, Sofia Panagiotidi, Javier Vázquez-Salceda, and Sanjay Modgil, *Towards a Formalisation of Electronic Contracting Environments*, Proceedings of the Workshop on Coordination, Organizations, Institutions and Norms -COIN@AAAI08-, at AAAI 08, Chicago, USA (2008).
- [Ost86] Elinor Ostrom, *An agenda for the study of institutions*, Public Choice (1986).

- [PANVS⁺08] Sofia Panagiotidi, Sergio Alvarez-Napagao, Javier Vázquez-Salceda, Nir Oren, Sandra Ortega, Roberto Confalonieri, Michal Jakob, Jiri Biba, Monika Solanki, and Steven Willmott, *Contracting Language Syntax and Semantics Specifications*, IST CONTRACT project (2008), 177.
- [Pas05] Adrian Paschke, *RBSLA A declarative Rule-based Service Level Agreement Language based on RuleML*, Computational Intelligence for Modelling (2005).
- [Pla68] Plato, *The Republic*.
- [Pör74] Ingmar Pörn, *Some Basic Concepts of Action*, Logical Theory and Semantic Analysis: Essays Dedicated to Stig Kanger on His Fiftieth Birthday, Springer, ISBN: 9027704384 (1974).
- [PTW05] Shamimabi Paurobally, Valentina Tamma, and Michael Wooldridge, *Cooperation and Agreement between Semantic Web Services*, W3C Workshop on Frameworks for Semantics in Web Services (2005).
- [PvdH06] Michael P Papazoglou and Willem-Jan van den Heuvel, *Service-oriented design and development methodology*, International Journal of Web Engineering and Technology (2006).
- [PVSAN⁺08] Sofia Panagiotidi, Javier Vázquez-Salceda, Sergio Alvarez-Napagao, Steven Willmott, and Roberto Confalonieri, *Intelligent Contracting Agents Language*, Proceedings of the Symposium on Behaviour Regulation in Multi-Agent Systems -BRMAS'08-, Aberdeen, UK (2008).
- [RA03] Juan Antonio Rodríguez-Aguilar, *On the Design and Construction of Agent-mediated Institutions*, PhD Thesis (2003).
- [RC92] James N Rosenau and Ernst Otto Czempiel, *Governance Without Government: Order and Change in World Politics*, Cambridge University Press (1992).
- [RL87] Cohen Philip R. and Hector J Levesque, *Persistence, Intention, and Commitment*, Storming Media (1987).
- [SC96] Filipe A. A Santos and José Carmo, *Indirect Action. Influence and Responsibility*, 1996, pp. 194–215.
- [Sco98] W Richard Scott, *Organizations: Rational, Natural, and Open Systems*, Prentice Hall (1998), 416.
- [Sco01] W Richard Scott, *Institutions and Organizations*, SAGE (2001), 255.
- [SH05] Munindar P Singh and Michael N Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*, Wiley (2005).
- [SHLP05] Marc-Thomas Schmidt, Beth Hutchison, Peter Lambros, and Rob Phippen, *The Enterprise Service Bus: Making service-oriented architecture real*, IBM Systems Journal (2005).
- [SJ06] Mårten Simonsson and Pontus Johnson, *Defining IT Governance-A Consolidation of Literature*, Proceedings of the 18th Conference on Advanced Information Systems Engineering (2006).
- [SJC97] Filipe A. A Santos, Andrew J I Jones, and José Carmo, *Action concepts for describing organised interaction*, System Sciences (1997).

- [Som06] I Sommerville, *Software Engineering*, Pearson Education, ISBN: 0321313798 (2006).
- [Spi89] J M Spivey, *The Z notation*, Prentice Hall International (UK) (1989).
- [ST95] Yoav Shoham and Moshe Tennenholtz, *On social laws for artificial agent societies: off-line design*, Artificial Intelligence (1995).
- [SW03] John Simpson and Edmund Weiner, *Oxford English Dictionary*, Oxford University Press (2003).
- [Tri04] Gerald Trites, *Director responsibility for IT governance*, International Journal of Accounting Information Systems (2004).
- [Tuo96] Raimo Tuomela, *Philosophy and distributed artificial intelligence: the case of joint intention*, John Wiley Sixth-Generation Computer Technology Series (1996).
- [VS04] Javier Vázquez-Salceda, *The Role of Norms and Electronic Institutions in Multi-Agent Systems: The HARMONIA Framework.*, PhD Thesis (2004).
- [VSAD04] Javier Vázquez-Salceda, Huib Aldewereld, and Frank Dignum, *Implementing Norms in Multiagent Systems*, G. Lindemann et al. (Eds.): MATES 2004, LNAI 3187, Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg (2004), 313–327.
- [VSAN08] Javier Vázquez-Salceda and Sergio Alvarez-Napagao, *Using SOA Provenance to Implement Norm Enforcement in e-Institutions*, Proceedings of the Workshop on Coordination, Organizations, Institutions and Norms - COIN@AAAI08-, at AAAI 08, Chicago, USA (2008).
- [VSANK⁺07] Javier Vázquez-Salceda, Sergio Alvarez-Napagao, Tamás Kifor, László Z Varga, Simon Miles, Luc Moreau, and Steven Willmott, *EU PROVENANCE Project: An Open Provenance Architecture for Distributed Applications*, R. Anicchiario, U. Cortés, C. Urdiales (eds.) Agent Technology and E-Health. Whitestein Series in Software Agent Technologies and Autonomic Computing. Birkhäuser Verlag AG, Switzerland, ISBN: 978-3-7643-8546-0 (2007), 55–64.
- [VSCP⁺03] Javier Vázquez-Salceda, Ulises Cortés, Julian Padget, Antonio López-Navidad, and F Caballero, *The organ allocation process: a natural extension of the Carrel Agent-Mediated Electronic Institution*, AI Communications (2003).
- [VSD03] Javier Vázquez-Salceda and Frank Dignum, *Modelling electronic organizations*, Multi-Agent Systems and Applications III (2003).
- [vW51] G H von Wright, *Deontic Logic*, Mind, New Series 60 (1951), no. 237, 1–15.
- [VWF02] Jens-S Vöckler, Michael Wilde, and Ian Foster, *The GriPhyN Virtual Data System*, Technical Report GriPhyN-2002-02 (2002).
- [WBC⁺08] Steven Willmott, Jiri Biba, Adam Ciganeck, Michal Jakob, Roberto Confalonieri, Dirk Bangel, Omar Sahyoun, and Sergio Alvarez-Napagao, *Web Services Framework for Contract Based Computing*, IST CONTRACT Project Deliverable (2008), 134.
- [web06] webMethods, *SOA Governance: Enabling Sustainable Success with SOA*, webMethods, Inc. (2006).

- [Wei04] Peter Weill, *Don't Just Lead, Govern: How Top-Performing Firms Govern IT*, MIS Quarterly Executive (2004).
- [WPMC⁺05] Steven Willmott, Félix Oscar Fernández Peña, Carlos Merida-Campos, Ion Constantinescu, Jonathan Dale, and David Cabanillas, *Adapting Agent Communication Languages for Semantic Web Service Inter-Communication*, Web Intelligence (2005).
- [WPR06] Phyl Webb, Carol Pollard, and Gail Ridley, *Attempting to Define IT Governance: Wisdom or Folly?*, Hawaii International Conference on System Sciences (2006).
- [WS97] Allison Woodruff and Michael Stonebraker, *Supporting Fine-Grained Data Lineage in a Database Visualization Environment*, Proceedings of the International Conference on Data Engineering (1997).
- [yLL02] Fabiola López y López and Michael Luck, *Towards a Model of the Dynamics of Normative Multi-Agent Systems*, Proceedings of the Int. Workshop on Regulated Agent-Based Social Systems (2002).
- [yLLd01] Fabiola López y López, Michael Luck, and Mark d'Inverno, *A framework for norm-based inter-agent dependence*, Proceedings of The Third Mexican International Conference on Computer Science (2001), 31–40.
- [ZLK⁺07] Yu Chen Zhou, Xin Peng Liu, Eduardo Kahan, Xi Ning Wang, Liang Xue, and Ke Xin Zhou, *Context Aware Service Policy Orchestration*, Web Services, 2007. ICWS 2007. IEEE International Conference on (2007), 936 – 943.

Publications

- [ANVS08] Sergio Alvarez-Napagao and Javier Vázquez-Salceda, *Using Provenance to implement a Norm Enforcement Mechanism for Agent-Mediated Healthcare Systems*, Proceedings of the Fifth Workshop on Agents Applied in Health Care at AAMAS'08, Estoril, Portugal (2008), 8.
- [ANVSK⁺06] Sergio Alvarez-Napagao, Javier Vázquez-Salceda, Tamás Kifor, László Z Varga, and Steven Willmott, *Applying provenance in distributed organ transplant management*, International Provenance and Annotation workshop (IPAW 2006), 3-5 May 2006, Chicago, USA, ISBN 978-3-540-46302-3 (2006).
- [CANP⁺08] Roberto Confalonieri, Sergio Alvarez-Napagao, Sofia Panagiotidi, Javier Vázquez-Salceda, and Steven Willmott, *A Middleware Architecture for Building Contract-Aware Agent-Based Services*, Proceedings of the International Workshop on Service-Oriented Computing: Agents, Semantics, and Engineering -SOCASE'08-, at AAMAS'08, Estoril, Portugal, ISBN 978-3-540-79967-2 (2008).
- [KVAN⁺06] Tamás Kifor, László Z Varga, Sergio Alvarez-Napagao, Javier Vázquez-Salceda, and Steven Willmott, *Privacy Issues of Provenance in Electronic Healthcare Record Systems*, First International Workshop on Privacy and Security in Agent-based Collaborative Environments (PSACE2006), Hakodate, Japan (2006).
- [KVVS⁺06] Tamás Kifor, László Z Varga, Javier Vázquez-Salceda, Sergio Alvarez-Napagao, Steven Willmott, Simon Miles, and Luc Moreau, *Provenance in Agent-mediated Healthcare Systems*, IEEE Intelligent Systems, ISSN 1541-1672-21 (2006), no. 6, 38–46.
- [PVSAN⁺08] Sofia Panagiotidi, Javier Vázquez-Salceda, Sergio Alvarez-Napagao, Steven Willmott, and Roberto Confalonieri, *Intelligent Contracting Agents Language*, Proceedings of the Symposium on Behaviour Regulation in Multi-Agent Systems -BRMAS'08-, Aberdeen, UK (2008).
- [VSAN08] Javier Vázquez-Salceda and Sergio Alvarez-Napagao, *Using SOA Provenance to Implement Norm Enforcement in e-Institutions*, Proceedings of the Workshop on Coordination, Organizations, Institutions and Norms - COIN@AAAI08-, at AAAI 08, Chicago, USA (2008).
- [VSANK⁺07] Javier Vázquez-Salceda, Sergio Alvarez-Napagao, Tamás Kifor, László Z Varga, Simon Miles, Luc Moreau, and Steven Willmott, *EU PROVENANCE*

Project: An Open Provenance Architecture for Distributed Applications, R. Annicchiarico, U. Cortés, C. Urdiales (eds.) *Agent Technology and E-Health*. Whitestein Series in Software Agent Technologies and Autonomic Computing. Birkhäuser Verlag AG, Switzerland, ISBN: 978-3-7643-8546-0 (2007), 55–64.

Glossary

- Abstract Norms, 57
- actor state p-assertion, 19
- administrative contract parties, 46
- anonymization, 26
- anonymization procedure, 28
- anonymized identifier, 28
- architecture governance, 60
- Artificial Social Systems, 9

- change-time governance, 63
- clause, 38
- communication manager, 48
- computation as interaction, 1
- context, 38
- context layer, 35
- contract, 33
- contract data model, 36
- contract instances, 49
- contract layer, 34
- contract manager, 48
- contract party, 36
- contract repository, 49
- contract templates, 49
- contract-aware agent-based services, 44
- contracting environment, 46
- contracting language, 34
- contractual electronic institution, 33
- Contractual Ontology, 41
- contractual ontology, 36
- corporate governance, 4
- corporation, 4
- Cremona, 33

- decision maker, 47
- deontic statement, 38
- design-time governance, 61

- dialogue manager, 49
- domain ontology, 42
- domain ontology layer, 34
- Dyadic Deontic Logic, 9
- Dynamic Logic, 9, 57, 58

- Electronic Institution or e-institution, 12, 55
- EU Provenance project, 19

- Flexible Normative Agents, 59

- governance, 3
- government, 3
- Grid, 3
- group list, 37

- Illocution, 13
- Institution, 11
- Institutional Theory, 6
- interaction p-assertion, 19
- interaction protocol layer, 35
- IST Contract project, 34
- IT governance, 4

- JINI, 3

- Kanger-Lindahl-Pörn Logical Theory, 9

- latent interaction, 24

- manager, 47
- message content layer, 35
- message layer, 35
- message manager, 49
- message transport, 66
- middleware, 44

- Monitoring, 17
Multi-Agent System, 1
- norm, 6
- OASIS Service Oriented Architecture Reference Model, 3
Objectives, 57
observer, 47
ontology store, 49
Open Gateway Service initiative, 3
- p-assertion, 19
PANDA, 32
performative, 43, 44
policy enforcement points, 65
predicate, 41
privacy, 27
process documentation, 23
protocol, 44
provenance, 18
Provenance Architecture, 19
provenance lifecycle, 20
provenance questions, 22
- registry, 63
registry/repository, 65
relationship p-assertion, 19
repository, 64
role enactment list, 36
Role hierarchy, 59
Roles, 59
RuleML, 33
run-time governance, 62
- Scene, 13
Service Oriented Architecture, 3
service-level agreement, 31
Service-level governance, 61
service-orientation, 2
SOA Governance, 60
SOA governance, 3
SOA governance system, 60
SOA lifecycle, 60
standard clause, 40
Standard Deontic Logic, 8
Statutes, 56
strongly connected process, 24
- Temporal Deontic Logic, 9
Theory of Rational Action, 8
- UDDI, 64
- Values, 56
Violation, 58
violation handling clause, 40
- W3C Web Services Architecture, 3
weakly connected process, 25
Web Service Level Agreement, 32
Web Services Interoperability, 61
workflow manager, 48
WS-Agreement, 32