Bringing social reality to Multiagent and Service architectures

Practical reductions for monitoring of deontic-logic and constitutive norms

Sergio Alvarez-Napagao

Advisor: Javier Vázquez-Salceda

PhD Thesis
Universitat Politècnica de Catalunya

Bringing social reality to Multiagent and Service architectures

Practical reductions for monitoring
of deontic-logic and constitutive norms

A thesis submitted for the degree of
Doctor of Philosophy in Artificial Intelligence
at the Polytechnic University of Catalonia

Tesi presentada per obtenir el títol de
Doctor en Intel·ligència Artificial
per la Universitat Politècnica de Catalunya

2015

Sergio Alvarez-Napagao

Advisor: Javier Vázquez-Salceda

PhD Thesis
Universitat Politècnica de Catalunya

Printed in Barcelona.

15 1

First edition: December 10, 2015

A Esther, por estar ahí siempre.

A mis padres, a los que les debo todo y, seguramente,
hasta que lean esto no se lo habrán acabado de creer.

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 mentally ill.
    - *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.

ORIGIN *late Middle English (sense 2 and sense 3): via Old French from Latin* institutio(n-), *from the verb* instituere *(see institute). Sense 1 dates from the early 18th cent.*

reality

noun

1. *the state of things as they actually exist, as opposed to an idealistic or notional idea of them:* he refuses to face reality | Laura was losing touch with reality.
    - *[count noun] a thing that is actually experienced or seen, especially when this is unpleasant:* the harsh realities of life in a farming community.
    - *[count noun] a thing that exists in fact, having previously only existed in one's mind:* we want to make the dream a reality.
    - *the quality of being lifelike:* the reality of Marryat's detail.
    - *[as modifier] relating to reality TV:* a reality show.
2. *the state or quality of having existence or substance:* youth, when death has no reality.
    - PHILOSOPHY *existence that is absolute, self-sufficient, or objective, and not subject to human decisions or conventions.*

ORIGIN *late 15th cent.: via French from medieval Latin* realitas, *from late Latin* realis 'relating to things' *(see* REAL*).*

New Oxford American Dictionary 3rd edition, *Oxford University Press, Inc., 2010.*

Short contents

Contents

List of Figures

List of Tables

Extended Abstract

As distributed systems grow in complexity, the interactions among individuals (agents, services) of such systems become increasingly more complex and therefore more difficult to constrain and monitor. We propose to view such systems as socio-technical systems, in which organisational and institutional concepts, such as norms, can be applied to improve not only control on the components but also their autonomy by the definition of soft rather than hard constraints.

Norms can be described as rules that guide the behaviour of individual agents pertaining to groups that abide to them, either by explicit or implicit support. The study of norms, and regulatory systems in general, in their many forms – e.g. social norms, conventions, laws, regulations – has been of interest since the beginning of philosophy, but has seen a lot of evolution during the 20th century due to the progress in the philosophy of language, especially concerning speech acts and deontic logic.

Although there is a myriad of definitions and related terminologies about the concept of norm, and as such there are many perspectives on how to analyse their impact, a common denominator is that norms constrain the behaviour of groups of agents in a way that each individual agent can build, with a fair degree of confidence, expectations on how each of their counterparts will behave in the situations that the norms are meant to cover. For example, on a road each driver expects everybody else to drive on only one side of the road (right or left, depending on the country). Therefore, normative contexts, usually wrapped in the form of institutions, are effective mechanisms to ensure the stability of a complex system such as an organisation, a society, or even of electronic systems. The latter has been an object of interest in the field of Artificial Intelligence, and it has been seen as a paradigm of coordination among electronic agents either in multi-agent systems or in service-oriented architectures.

In order to apply norms to electronic systems, research has come up with abstractions of normative systems. In some cases these abstractions are based on regimented systems with flexible definitions of the notion of norm, in order to include meanings of the concept with a coarse-grained level of logic formality such as conventions. Other approaches, on the other hand, propose the use of deontic logic for describing, from a more theoretical perspective, norm-governed interaction environments. In both cases, the purpose is to enable the monitoring and enforcement of norms on systems that include – although not limited to – electronic agents. In the present dissertation we will focus on the latter type, focusing on preserving the deontic aspect of norms.

Monitoring in norm-governed systems requires making agents aware of: 1) what their normative context is, i.e. which obligations, permissions and prohibitions are applicable to each of them and how they are updated and triggered; and 2) what their current normative status is, i.e. which norms are active, and in what instances they are being fulfilled or violated, in order words, what their social – institutional – reality is.

The current challenge is on designing systems that allow computational components to infer both the normative context and social reality in real-time, based on a theoretical formalism that makes such inferences sound and correct from a philosophical perspective. In the scope of multi-agent systems, many are the approaches proposed and implemented that fulfill these requirements up to this date. However, the literature is still lacking a proposal that is suited to the current state-of-the-art in service-oriented architectures, more focused nowadays on automatically scalable, polyglot amalgams of lightweight services with extremely simple communication and coordination mechanisms – a trend that is being called *microservices*.

This dissertation tackles this issue, by 1) studying what properties we can infer from distributed systems that allow us to treat them as part of a socio-technical system, and 2) analysing which mechanisms we can provide to distributed systems so that they can properly act as socio-technical systems. The main product of the thesis is therefore a collection of computational elements required for formally grounded and real-time efficient understanding and monitoring of normative contexts, more specifically:

1. An ontology of events to properly model the inputs from the external world and convert them into brute facts or institutional events;
2. A lightweight language for norms, suitable for its use in distributed systems;
3. An especially tailored formalism for the detection of social reality, based on and reducible to deontic logic with support for constitutive norms;
4. A reduction of such formalism to production rule systems; and
5. One or more implementations of this reduction, proven to efficiently work on several scenarios.

This document presents the related work, the rationale and the design/implementation of each one of these elements. By combining them, we are able to present novel, relevant work that enables the application of normative reasoning mechanisms in real-world systems in the form of a practical reasoner. Of special relevance is the fact that the work presented in this dissertation simplifies, while preserving formal soundness, theoretically complex forms of reasoning. Nonetheless, the use of production systems as the implementation-level materialisation of normative monitoring allows our work to be applied in any language and/or platform available, either in the form of rule engines, ECA rules or even if-then-else patterns.

The work presented has been tested and successfully used in a wide range of domains and actual applications. The thesis also describes how our mechanisms have been applied to practical use cases based on their integration into distributed eldercare management and to commercial games.

Resumen

Con el incremento en la complejidad de los sistemas distribuidos, las interacciones entre los individuos (agentes, servicios) de dichos sistemas se vuelven más y más complejas y, por ello, más difíciles de restringir y monitorizar. Proponemos ver a estos sistemas como sistemas socio-técnicos, en los que conceptos organizacionales e institucionales (como las normas) pueden aplicarse para mejorar no solo el control sobre los componentes sino también su autonomía mediante la definición de restricciones débiles (en vez de fuertes).

Las Normas se pueden describir como reglas que guian el comportamiento de agentes individuales que pertenecen a grupos que las siguen, ya sea con un apoyo explícito o implícito. El estudio de las normas y de los sistemas regulatorios en general y en sus formas diversas – normas sociales, convenciones, leyes, reglamentos– ha sido de interés para los erudítos desde los inicios de la filosofía, pero ha sufrido una evolución mayor durantre el siglo 20 debido a los avances en filosofía del lenguaje, en especial los relacionados con los actos del habla –speech acts en inglés– y formas deónticas de la lógica modal.

Aunque hay una gran variedad de definiciones y terminología asociadas al concepto de norma, y por ello existen varios puntos de vista sobre como analizar su impacto, el denominador común es que las normas restringen el comportamiento de grupos de agentes de forma que cada agente individual puede construir, con un buen nivel de confianza, expectativas sobre como cada uno de los otros actores se comportará en las situaciones que las normas han de cubrir. Por ejemplo, en una carretera cada conductor espera que los demás conduzcan solo en un lado de la carretera (derecha o izquierda, dependiendo del pais).

Por lo tanto, los contextos normativos, normalmente envueltos en la forma de instituciones, constituyen mecanismos efectivos para asegurar la estabilidad de un sistema complejo como una organización, una sociedad o incluso un sistema electrónico. Lo último ha sido objeto de estudio en el campo de la Inteligencia Artificial, y se ha visto como paradigma de coordinación entre agentes electrónicos, tanto en sistemas multiagentes como en arquitecturas orientadas a servicios.

Para aplicar normas en sistemas electrónicos, los investigadores han creado abstracciones de sistemas normativos. En algunos casos estas abstracciones se basan en sistemas regimentados con definiciones flexibles del concepto de norma para poder incluir algunos significados del concepto con un menor nivel de granularidad formal como es el caso de las convenciones. Otras aproximaciones proponen el uso de lógica deóntica para describir, desde un punto de vista más teórico, entornos de interacción gobernados por normas. En ambos casos el propósito es el permitir la monitorización y la aplicación de las normas en sistemas que incluyen –aunque no están limitados a– agentes electrónicos. En el presente documento nos centraremos en el segundo tipo, teniendo cuidado en mantener el aspecto deóntico de las normas.

La monitorización en sistemas gobernados por normas requiere el hacer a los agentes conscientes de: 1) cual es su contexto normativo, es decir, que obligaciones permisos y prohibiciones se aplican a cada uno de ellos y cómo se actualizan y activan; y 2) cual es su estado normativo actual, esto es, que normas están activas, y que instancias están siendo cumplidas o violadas, en definitiva, cual es su realidad social –o institucional–.

En la actualidad el reto consiste en diseñar sistemas que permiten inferir a componentes computacionales tanto el contexto normativo como la realidad social en tiempo real, basándose en un formalismo teórico que haga que dichas inferencias sean correctas y bien fundamentadas desde el punto de vista filosófico. En el ámbito de los sistemas multiagente existen muchas aproximaciones propuestas e implementadas que cubren estos requisitos. Sin embargo, esta literatura aun carece de una propuesta que sea adecuada para la tecnología de las arquitecturas orientadas a servicios, que están más centradas en amalgamas políglotas y escalables de servicios ligeros con mecanismos de coordinación y comunicación extremadamente simples, una tendencia moderna que lleva el nombre de microservicios.

Esta tesis aborda esta problemática 1) estudiando que propiedades podemos inferir de los sistemas distribuidos que nos permitan tratarlos como parte de un sistema socio-técnico, y 2) analizando que mecanismos podemos proporcionar a los sistemas distribuidos de forma que puedan actuar de forma correcta como sistemas socio-técnicos. El producto principal de la tesis es, por tanto, una colección de elementos computacionales requeridos para la monitorización e interpretación eficientes en tiempo real y con clara base formal. En concreto:

1. Una ontología de eventos para modelar adecuadamente las entradas del mundo exterior y convertirlas en hechos básicos o en eventos institucionales;
2. Un lenguaje de normas ligero y sencillo, adecuado para su uso en arquitecturas orientadas a servicios;
3. Un formalismo especialmente adaptado para la detección de la realidad social, basado en y reducible a lógica deóntica con soporte para normas constitutivas;
4. Una reducción de ese formalismo a sistemas de reglas de producción; y
5. Una o más implementaciones de esta reducción, de las que se ha probado que funcionan eficientemente en distintos escenarios.

Este documento presenta el estado del arte relacionado, la justificación y el diseño/implementación para cada uno de esos elementos. Al combinarlos, somos capaces de presentar trabajo novedoso y relevante que permite la aplicación de mecanismos de razonamiento normativo en sistemas del mundo real bajo la forma de un razonador práctico. De especial relevancia es el hecho de que el trabajo presentado en este documento simplifica formas complejas y teóricas de razonamento perservando la correctitud formal. El uso de sistemas de reglas de producción como la materialización a nivel de implementación del monitoreo normativo permite que nuestro trabajo se pueda aplicar a cualquier lenguaje o plataforma disponible, ya sea en la forma de motores de reglas, reglas ECA o incluso patrones si-entonces.

El trabajo presentado ha sido probado y usado con éxito en un amplio rango de dominios y aplicaciones prácticas. La tesis describe como nuestros mecanismos se han aplicado a casos prácticos de uso basados en su integración en la gestión distribuida de pacientes de edad avanzada o en el sector de los videojuegos comerciales.

Acknowledgments

'Nothing's too good for the man who shot Liberty Valance.'

The Man Who Shot Liberty Valance
John Ford

Behind this dissertation there is a *very* long journey. On one hand, at some points it has looked like it was going to be never-ending; on the other hand, it truly has been a remarkable experience. After all, as they say, the journey is more important than the destination – and even if this is not true, I will be happy to buy it anyway.

Actually, these might not seem the best times for devoting years and more years to research but, nonetheless, I can hardly regret anything. I have learnt more than I would have ever expected and I have had to fight battles that I am proud to have fought. But, more importantly, I have met, worked with, and shared the journey with quite a lot of amazing people who, in the end, are the ones that give true meaning to everything else.

First and foremost, I cannot express my gratitude enough to my advisor, Javier Vázquez-Salceda. Since the moment that he trusted me for a difficult task in a research project, he has been a constant push for me for evolving as a researcher, providing moral support when it was needed. Above all, I really have to appreciate his patience, his always accurate insights and his ethics.

The Knowledge Engineering and Machine Learning group (KEMLg) has been my home for all these years. Special thanks have to go to Ulises Cortés, who has always been a constant spur, challenging conceptions and continuously proposing fascinating applications and side projects; and to Ramon Sangüesa, who let me embark an ambitious project while I still was an undergraduate student and taught me a lot of what research is and what research means. I also have to thank Steve Willmott, who set the foundations of some of the contributions of this dissertation and from whom I learnt about the balance between research and practical applications. Also, thanks to the rest of the *seniors* of the KEMLg, they where always there when their help was needed, especially Karina Gibert, Miquel Sánchez and Javier Béjar.

I have had the opportunity to work in challenging projects closely with people that I can feel really fortunate to have worked with. João, many thanks for so many conversations, dinners, trips, crazy projects. I really miss them. Roberto, Luis, Ignasi, Arturo, thanks for being there in the good and the bad moments, and for always providing enlightenment over so many topics. And also for the fun moments, which have been many. Sofia, without your help in that last push, trying to make some kind of sense out of so many whiteboards full of formulas, this dissertation would not be the same, not even close. I learned a lot from you, and I will always appreciate having shared part of this journey.

Also, I want to express my gratitude to the rest of people of the office. We have shared many great moments and we have had countless challenging debates. I really miss being there. Darío, Bea, Soraya, Jonathan, Miquel, Carlos, and all the rest, thanks a lot. And I cannot forget about Mercè Juan, who has taken care of my administrative status much more than I deserved.

I would like to express my gratitude for my hosts during my stay in Utrecht: Frank Dignum, Virginia Dignum and Huib Aldewereld, you really helped me feel at home. Every debate with you was enlightening, and thanks to the stay I could finally lay out the foundations of this thesis.

I have also had the opportunity to travel a lot and to meet and work with brilliant people that have been a source of constant knowledge and in many instances have provided support for the good ideas or a challenge for the bad ones. The list would be countless, but I want to mention my appreciation for Pablo Noriega, Julian Padget, Michael Luck, Wamberto Vasconcelos, Fernando Koch, Nir Oren, Felipe Meneguzzi, Simon Miles, Sanjay Modgil, Kifor Tamas and Owen Cliffe. I am sure I am missing a lot of names, but I am getting old and my memory starts to fail.

My sanity check during these years has been filmmaking, most of which I could make in AVED at the campus. I have to consider myself truly fortunate to have met, worked, created, travelled and shared so many beers, meals and unforgettable experiences with Javier, Carme and Jordi. I cannot be more grateful to have been given the opportunity to help in such creative projects and to be part of such great teamwork. Jordi, Marc, Silvia, Iban, Carlos, Jonathan, Pau, Sergi, Pilar, Montse, and so many others: many thanks for being such great companions in this hobby that became so much more.

I also have to deeply thank a lot of people I have not been able to see as often as I should have during the last few years. Albert, Cris, Ramon, Miguel, kampde, Zeus, and many others, I thank you for your patience and for the great moments we have lived together. The same goes for Jordi, JL and Jesús, with whom I made an amazing trip I will never forget.

I am forgetting to mention a lot of people that deserved it, I am sure of that, but it is not intentional. I hope that everybody that notices this will forgive my omissions.

Sergio Alvarez-Napagao
London
December 2015

Introduction

Butch: 'What happened to the old bank? It was beautiful.'
Guard: 'People kept robbing it.'
Butch: 'Small price to pay for beauty.'

Butch Cassidy and the Sundance Kid
George Roy Hill

In the last decades Information and Communication Technologies have reshaped not only the way information is processed and accessed but also the way people use this information and, more recently, the way they interact with others. From the technological point of view, technical advances such as the Internet have made computation evolve from the single/mainframe computer (where computing is a process driven by the computer's CPU) into computer networks (where computing is driven by the interaction between distributed computational units). From the social point of view, ICT have transitioned from being a tool to foster productivity in a company to a mediator in social relations, up to its current state where technical developments inspire new forms of social interaction [Whitworth 2009].

Since the early 60's researchers have studied the interaction between technology systems and social systems. The term Socio-Technical System (STS), coined by Eric Trist, Ken Bamforth and Fred Emery, was originally applied to study how the interaction of people with technology may affect the overall performance of companies. But soon the term was expanded to the study of the interaction of any type of social system with a technological one, with a focus on the social benefit a well-structured STS may bring. The rise and expansion of Internet in the early 90's has been of special interest in socio-technical studies, especially to analyse the shift on how people use it: from the Internet seen as a way to host information to the pervasive networking technologies which host social interactions [Whitworth 2009]. Currently STS are a hot topic of research, creating new terms such as *virtual socio-technical design*, *socio-technical communities* and *socio-technical societies*.

An interesting idea coming from socio-technical studies is that, as technology becomes part of social life, social aspects should be taken into account in the design of technology. This is of special relevance in current ICT technologies. In the last years we are in the middle of a rapid increase in distributed systems based on small services –

and APIs – that carry out tasks ranging from simple mathematical calculations to operations that have an impact on the physical world, (e.g. online shopping). Furthermore, some technologies such as the Internet of Things (IoT) are challenging our current definitions on the boundaries between the physical and the virtual worlds.

On top of that, Artificial Intelligence techniques in general and Multiagent Systems (MAS) in particular may bring STS to a new level where the *human-computer divide* is not absolute, and technology is not only supporting human societies but actively participating in mixed societies composed by both human and artificial entities. Under this new vision on STS, artificial entities can be modeled as computer agents (both software agents and embodied, physical agents such as robots). New challenges raise from this vision, as these computer agents situated in a mixed human/artificial society must model social goals, responsibilities and dependencies in order to properly participate in such society [Kolp and Wautelet 2011].

How should artificial entities behave in such mixed human/artificial societies? An example of system of the current state of the art is Watson, an IBM software capable of answering a range of questions so wide that it has been able to win in the pop-quiz-like contest *Jeopardy*, broadcasted by the US channel *NBC*. Although from a technological point of view Watson is not intended as a socio-technical system but as a proof of the effectiveness of data-processing systems, it is still a pertinent example because of its public impact and due to its apparent capabilities to behave like a part of a human institution (a contest).

Watson winning in this contest is particularly relevant because it played with exactly the same rules that are applied to the rest of the contestants: the questions were provided in natural language and they could belong to any topic of pop culture – sports, literature, art, science, language, etc. Under these conditions, the game is especially complex for a machine but Watson, nonetheless, was capable of correctly interpreting the answers and identifying the correct context for an 88% of the questions allowing it to win more money than its human counterparts.

On the other hand, as is also the case with any machine learning algorithm (including the current trends in deep learning) the reasoning capabilities of such systems are strongly biased and even determined by the inputs fed to them. Pop culture questions can be trained by feeding magazines, books, encyclopedias and transcripts. Which is nothing short of impressive and useful anyway, because Watson is already proving its utility in other fields, especially in healthcare.

Aside from what they are trained for by statistically analysing such inputs, it is obvious that there are still too many questions that Watson would not be able to answer, especially those related with its identity as an agent. Inquiries related to conciousness are among the most immediate examples that might come to mind, but that is still too much of a complex topic to discuss, and there is not even a scientific consensus on a proper definition.

But we do not need to delve so far to find other kinds of questions that would seriously challenge Watson. For example, *what is the role I have inside Jeopardy?* Or: *what are the rights that I am given by being what I am? And which obligations do I have for the same?*

And outside of Jeopardy?

Watson would not be able to answer these questions because it has neither been designed nor programmed to do so. And a sensible answer to this issue would be that those are irrelevant questions because that is not the purpose of this computational system. However, it is being *treated* as a human inside the contest: during the contest, no human is operating it, the same rules that apply to any other contestant still apply to it, and humans are competing against it, winning or losing real money. So, in a way, it makes sense that we ask ourselves about the relationship between Watson and the other people involved in Jeopardy and, by extension, it also makes sense that we ask ourselves whether it is possible that the machine can also ask itself about the same concerns. Because these concerns are not at all irrelevant: how would Watson act, without being reprogrammed, if the rules changed in the middle of the game? Evidently, it would probably fail at playing under the new set of rules. In other words, it would have a erratic or irrational behaviour if we take into account how a human would act.

Humans, after all, and as a distinctive feature compared to the rest of the animals, are capable of interpreting the physical reality in terms of conventions or rules far beyond of what exists in the physical world or whatever information is genetically or physiologically coded in their mind. This is not limited to interpretation: groups of people, and societies by extension, can create and assign (commonly understood) meanings to objects, and derivations of combinations of such objects, that were not primarily designed with that intention, as long as there is a common, accepted understanding that such assignments of meaning are valid.

An example is the pervasive use of *emoticons* in the Internet. The colon (:) and the closing bracket ()) were created as punctuation symbols, but when put together (:)) they are, as a derived combination, commonly interpreted as a smiling face. This interpretation is not truly original, but rather has been used in similar ways in some media formats such as comics and cartoons. Two points can be interpreted as two eyes, and a curve can be interpreted as a smiling mouth, and therefore the combination of both can be interpreted as a smiling face.

This is possible because, and only because, society has, more or less progressively, adopted such kind of interpretations as commonly accepted. (Almost) nobody would use emoticons if there was not a common agreement on their meaning.

This has already been the topic of ongoing research on semiotics and other areas of knowledge. Actually, the interpretation of images from abstract forms to the level of concepts is already being tackled by neural networks and deep learning algorithms. However, at some point in the workflow a human or group of humans is always required, directly or indirectly, in order to label the *commonly accepted* meaning of the detected concepts.

One can argue at this point that such acquired meanings are sort of obvious due to physical similarity to the reality projected by these symbols. However, there are examples that raise the complexity to limits that are unbearable at first sight. Take money as example, in any of its forms: gold, coins, notes, cheques, fiat money, electronic money. Is this not a much less obvious interpretation between each of these forms and what money really means for us?

A bank note is, physically, no less and no more, a piece of paper with some drawings and some writing. However, with a bank note we are granted the possibility of taking things from shops up to a certain *value*. Therefore, a bank note is interpreted by society as granting a certain power, in other words, it is commonly accepted that it has a meaning that is deontic in nature. Interestingly, electronic money, which is virtual rather than physical, due to temporarily having no physical grounding, receives the same treatment. In summary, society can grant deontic powers to physical or abstract objects.

Money has its own rules, accepted by society. Once it is accepted that something counts as money, many dispositions and *rules of play* come into action. One can take something in exchange of money, lend it to other people, and so on. That is, a set of *norms* apply to people that have money. With a set of norms, it can be said that there is an *institution*. Even if money changes, such as the conversion from Pesetas to Euros, or if new types of money are introduced, as electronic money in its moment, the basic norms that apply to money tend to persist.

Society handles, and in fact is continuously founded and evolving on top of, a vast amount of institutions: money, marriage, academia, language, and so on, and in order to understand social reality, we need to interpret human interaction in the context of these institutions. Institutions are social mechanisms supported by norms that constraint the behaviour of the individuals belonging to them. Usually, these norms are created as a means to provide welfare to these individuals. Sometimes they work as they should, most of the times they do not, but their objective is to minimise conflicts, to reduce uncertainty, to force some predictability and provide safety on the interactions between people.

As small as a family or as large as a country, an institution causes relevant effects on the individuals and even on other institutions. A person might be happy to belong to it, or be prosperous taking profit of the opportunities it provides. On the other side, a person might cause damage while defending it, or be a victim of its power. Institutions are part of our everyday life, giving shape to societies and playing a main role in the progress of humanity. In short, we are all affected by them and, at the same time, we all try to get profit of belonging to them, even at the expense of the actual institutional objectives. Sometimes we even try to change them.

Institutions are also the building block on top of which *organizations* are built. Organizations, which are social structures enabling roles to be enacted and objectives to be followed, use the mechanisms provided by institutions in order to function correctly. Organizations have no success guaranteed by definition, no matter how well the institutional rules of the game are designed, or how well the participants of the organization fit its roles, or how simple or easy are the organizational objectives. *Governance* mechanisms are always desirable, as a means to audit the performance of the organization.

The process of governance involves dealing with practices and methodologies that seek for a better control and management over the life-cycle of an organization in order to optimise the outcomes of its participants. Usually, governance is implemented by *enforcement* of the organizational rules, including institutional norms and subsequent analysis of the performance against predefined metrics. That is, if the members of the organization can rely on the fitness of the rules to fulfil the objectives, then we can say that they have the responsibility for these rules to be *observed* for both collective and individual benefit. When rules are observed, governance brings about the stability, predictability that makes participants trust institutional mechanisms on their own benefit.

Enforcement and observance are, therefore, two critical processes of organizations and, by extension, societies. It can be the case and, as said before, it is more common than not, that norms are not really optimal from a collective or individual point of view. But norms can be improved, and if we are to suppose that norms are set in good faith, then consequently both enforcement and observance have to be prepared and executed.

Both concepts are, in fact, quite intertwined. There can be no observance without enforcement, as rules that cannot be put in order, there is no way to check their compliance. Moreover, there can be no enforcement without observance, as rules that cannot be checked will not be able to guarantee trust between participants.

And now we are in disposition to close the circle that began with apparently extraneous topics such as interpretation and common understanding of symbol and object meanings. In order for a rule to be observed, we need to have the proper apparatus to understand how things that are, things that happen, and things that are produced by participants, *mean* something that is relevant for a particular rule.

This dissertation, built on top of theoretical concepts related to electronic institutions, develops formal artifacts, and their reduction to software components, that enable distributed systems, in the broad sense of the term, to understand norms and in consequence process the physical reality in order to perceive social reality in real-time. This is a necessary but not sufficient condition for a software component to interpret their role in society[1] and, reciprocally, let others see the electronic system as a role enactor in their society.

All these concepts that have appeared in this discourse will be properly discussed in Part I. In the next section, we will present which are the research questions that this document will try to tackle, and in order to do that we will base such research questions on the theoretical background that we have just presented. In summary: we have seen that there are social mechanisms that allow for assigning meanings to physical or virtual objects or actions that are commonly understood; that these meanings can involve deontic notions and as such they can be the building blocks of institutions through the establishment of norms; that groups of people can use the mechanism offered by institutions to create organizations with common objectives; that governance is a desirable feature of such organizations, involving enforcement and observance; and that in order to have observance, we need to understand the social mechanisms that give institutional meaning to what happens in reality.

[1]From now on we will use the term *society* to refer to a kind of socio-technical society composed by humans and artificial entities, as described in [Kolp and Wautelet 2011]. And we will use the term socio-technical system or STS to refer to the joint of the social and technical dimensions in such societies.

Research questions

As discussed in the previous section, the objective of this thesis is to provide mechanisms to distributed systems to enable the perception of social reality from inputs coming from physical reality. A first concern on this respect is, therefore, to identify what kind of electronic systems should, and which are able to, integrate such mechanisms:

Research Question 1 *What properties can we infer from individuals in distributed systems that allow us to treat them as part of a socio-technical system?*

An answer to this will help us categorise the elements of a distributed system. From this, the question is:

Research Question 2 *What mechanisms can we provide to distributed systems so that they can properly act as socio-technical systems?*

This last question will produce a list of formal and practical components that will be the core of our contributions.

Methodology and contributions

One of our main assumptions is considering a broad meaning of the term distributed system. Our objective is to include not only intelligent agents, which are usually the computational element used in similar research, but also service-oriented architectures and even the current trend of microservices.

By the study of the properties of such systems with respect to agency, we choose norms and institutions as the proper abstraction to apply. Our objective along the document will be to create **a computer-readable language based on norms** to those used by humans. This will be our first contribution, and it will allow our electronic systems to work with a vocabulary capable of interpreting their *social reality*.

However, detection of social reality is not trivial due to its dependence on logic formalisms such as deontic logics. We will propose **a reduction from complex logics to rules of production systems**, reducing such complexity at the same time that we will be able to inject those rules in as many programming languages as possible, as we are targetting services.

The final contribution of this thesis will be **a monitor capable of discovering the social reality of an agent in a distributed system**. This contribution will build upon our previous contributions: language and formal reduction.

Our proposal is that the adoption of this kind of components to process social reality can bring about a wide range of applications that benefit from institutional abstractions.

Structure of this document

This document is divided in three major parts.

The first one includes the theoretical grounding of the conceptual background on top of which the rest of document is built upon, and is enclosed in Part I. Chapter 1 distinguishes between the possible implementations of distributed systems. These implementations are analysed form an agency perspective in Chapter 2 and we propose norms as an abstraction of individual constraint for distributed systems participants. Then we analyse the use of systems based in norms for collective constraints in Chapter 3.

The central part, Part II contains our contributions. These are a lightweight language for norms for computational components (Chapter 5), operational semantics for this language (Chapter 6), and a translation from these operational semantics to practical implementations (Chapter 7).

Finally, in Part III we wrap things up. First, we study the relationship between our contributions and enforcement mechanisms (Chapter 8), we demonstrate a practical use case to demonstrate our contributions based on their integration in commercial games (Chapter 9), and we end up with general conclusions (Chapter 10).

PART I

STATE OF THE ART

CHAPTER 1

Computation as interaction

He'd understood then why Wintermute had chosen the nest to represent it, but he'd felt no revulsion. [. . .] Wintermute was hive mind, decision maker, effecting change in the world outside.

Neuromancer
William Gibson

Nowadays, communication over the Internet is of utmost relevance, not only for human-to-human interaction, but also between individuals and businesses. In both cases, the use of servers that act with variable degrees of automation and autonomy have become pervasive: banking, shopping, education, public administrations, and so on.

Therefore, computation in many areas of interest has evolved from mere mechanisms for data storage and processing and information retrieval into semi-automated networks of computers. This new metaphor has received the name of *computation as interaction* [Luck et al. 2005]. In this paradigm of computation, computing becomes a social activity. Social in the sense that the results of such computing are the result of a joint action between several *agents* – human agents or software agents. Investigating about what an agent is and is not, and what to be an agent implies in terms of the relationship to other agents, is very important in order to explore both how social reality can be established on top of distributed computational architectures and how governance mechanisms can be introduced in order to ensure an acceptable level of social order.

In this chapter, we will discuss about the concept of agency, which will allow us to understand the individual components of such social activities, and we will explore different types of architecture that have been used to materialise this *computation as interaction* paradigm as software distributed systems.

1.1 Agency and agents: distributed systems

What constitutes an agent? Is an agent pressupossed to have free will? Is an agent merely any rational being? Is it desirable to control or constrain the freedom of other agents? It is easy to see that agency is a concept that is related to many branches of philosophy, such as moral or metaphysics, and therefore it has been the subject of much debate since Plato and Aristotle.

In our case, we are more interested in a purely analytical definition, because our focus is on governance systems in which we assume the existence of norms that are not, or at least should not be, affected by moral implications. Such a definition can be found in [Davidson 1971]: *"a person is an agent of an event if and only if there is a description of what he did that makes true a sentence that says he did it intentionally"*. An agent is exercising agency, thus, if and only if the instantiation of certain mental states and events (beliefs, desires, intention) are able to produce the right (possibly physical) events, generally in the form of a chain of causal relations [Davidson 2002].

If agency is related to mental states, does that mean that agency is restricted to humans (and animals) or it can be applied to artificial systems as well? If so, can we consider that artificial systems can have *mental* states? These are fairly contentious and still open topics of discussion, but some researchers have empirically shown that agency does not necesarily entail conciousness [Bargh et al. 2001]. Furthermore, [Dennett 1989], following an instrumentalist perspective and turning the question upside down, argues that predictive patterns of behaviour supporting complex goals inevitably indicate the existence of mental states, because such existence cannot be separated from the existence of agency.

These arguments support the idea of artificial systems capable of agency. An example of formalisation of this concept is the Belief-Desire-Intention (BDI) agent [Rao and Georgeff 1995]: a non-physical system, grounded on multi-modal logic, that is capable of storing and transforming information, motivational and deliberative states (*mental attitudes*) and with the capability to *act* by selecting and executing appropriate actions or procedures in order to achieve goals.

BDI agents are not the only artificial systems that have this capability of acting. Nowadays Internet is immersed in a myriad of countless web-services and application programming interfaces (APIs) that not only digest and provide information but also trigger complex actions and processes that have physical implications (e.g. online shopping), from simple inputs. We will assume in this dissertation, following the thesis in [Dennett 1989], that web-services and APIs are capable of agency, at least at the same level as BDI agents.

All agents, be them humans (animals) or artificial systems, are rarely isolated. Almost no agent is self-sufficient and generally agents must communicate and coordinate among themselves in order to fulfil individual or collective objectives. This is true for humans as well as for artificial systems, although it is important to note that artificial systems are built in a coordinated (thus distributed) fashion, in virtually every case, to produce information or functionality to a human end-user: *computation as interaction*.

In the following sections we will discuss several relevant abstractions of coordination structures that have been created in order to fulfil this paradigm.

1.2 Agent orientation and Multi-agent systems

Agent-oriented computing appeared in the early 90s with the notion of an *intelligent agent* as the proposal of a key foundation for distributed systems. An intelligent agent is, in this context and explained briefly, an autonomous entity realised as a software component, capable of making its own proactive decisions with no external control, as well as capable of processing and reacting to perceptions from the environment [Wooldridge and Jennings 1995]. Intelligent agents are also able to communicate with other agents to share information and to coordinate join actions in order to achieve complex goals.

A group of agents that interact among themselves to manage their – goal or task – dependencies, sometimes sharing some common objectives is called Multiagent System (MAS).

All these attributes suggest that multi-agent systems are well-suited to solve complex problems where coordination among several entities is valuable in order to find an optimal – or near to optimal – solution with limited time and bounded computational resources. Intelligent agents may also exhibit other useful attributes such as *learning, rationality, mobility, benevolence* and *veracity*.

Examples of Multi-agent frameworks are (but are not limited to): JADE [Bellifemine, Poggi, and Rimassa 2001], Jack [Winikoff 2005], 2-APL [Dastani 2008], Jason and AgentSpeak [Bordini and Hübner 2005].

Research on MAS has been mainly academic, and therefore intelligent agents have been generally designed with strong formal groundings, i.e. multi-modal logic-based reasoning, communication based on philosophy of language, coordination mechanisms based on organisational theory. These attributes make MAS capable of distributing work in an adaptive way, allowing in some cases dynamic re-organisation. Thanks to this, MAS gained commercial success in many fields of information technology, from computer networking to decision support systems [Luck et al. 2005].

More specifically, there are three features that are of special relevance to analyse the benefits of MAS when building distributed systems: speech act theory, communication languages, and interaction protocols. The following subsections briefly explain them.

1.2.1 Speech Act Theory

Speech Act Theory is the area of the philosophy of language that studies the pragmatics[1] of the different types of messages that two agents can exchange [Searle 1969]. This theory proposes that certain uses of language can create acts that have an influence in the external world: *saying does something* [Austin 1975].

For example, if a person says that he promises to do something, the actual saying is the promising. In saying, he is committing to a promise. In this context, promising is a performative utterance, distinguished from other kinds of utterances as for example

[1]The study of how language is used by people in order to achieve their intentions.

"today is raining". As a consequence, everything that is uttered has been uttered with the intention of satisfying a certain goal.

Every act regarding an utterance has several components:

- Locution: the meaning of the statement itself
- Illocution: the contextual function of the act (intended by the utterer)
- Perlocution: the results of the act upon the listener

Speech acts can be categorised in several types: representative (merely informative), directives (enforcing other individual to perform an action), commisives (committing to perform an action), expressives (expressing emotions or feelings), and declarations (changing the state of the world) [Searle 1985].

A speech act is defined as the conjunction of a performative verb and a propositional content. The former contains an illocutionary force [Levinson 1983], e.g. request, inform, promise; while the latter is the object of the illocutionary force. For example, in the speech act *"I declare you husband and wife"*, the performative verb would be *declare*, while the propositional content would be *you be husband and wife*.

Speech acts have been widely used in Multi-agent frameworks as a fundamental piece of the design of artificial languages for agent communication.

1.2.2 Agent communication languages

The main motivation to have languages specifically tailored for agents is to allow meaningful interactions between heterogeneous agents. Agents involved in a communicative process should be able to exchange information and knowledge with the minimum risk of misunderstanding possible.

Following Speech Act Theory, the agent communication languages (ACLs) make a distinction between the intentional part of the message (the message container) and the propositional content (the message body). The intentional part is usually described by means of performatives, and each ACL defines its own set of valid performatives and semantics, allowing receivers to interpret the illocutionary force involved in the communicative act.

Therefore, ACLs help shaping messages in a way that not only the object of the message is understood, but also the intentions with respect to it. Examples of ACLs include: KIF [Genesereth and Fikes 1992], KQML [Finin et al. 1994] and FIPA-ACL [Agents 2002a].

1.2.3 Agent communication protocols

While Speech Act Theory and ACLs give form to a more expressive means of communication among agents, agent communication protocols define the behavioural aspect of the communication.

A set of agent communication protocols (also referred to as interaction protocols) define *acceptable* sequences of messages in specific interaction contexts, e.g. a query from agent A to agent B, or a negotiation between an arbitrary number of parties. Protocols provide a standard for interaction by pre-defining the way interactions should be structured and orchestrated.

An individual protocol defines, for a given message in a certain interaction context, which messages can be sent next. This reduces the burden on both the message sender and receiver, as for each message there are guidelines on how to respond. Because protocols are modelled in a structural way, parties can track the status of each party and its conversations in the interaction context. Agent communication protocols are analogous to network protocols, with the addition of speech acts as first-order objects.

The most common set of agent communication protocols used is the FIPA-IP specification [Agents 2003], which defines the following protocols: *request, query, request_when, contract_net, iterated_contract_net, english_auction, dutch_auction, brokering, recruiting, subscribe* and *propose*.

One of the limitations of interaction protocols such as the ones defined in the FIPA specification is the fact that they are structural rather than based on clear semantics [Dignum et al. 2007]. This makes it harder for agents to reason about them. Several solutions have already been proposed in the literature [Eijk, Huget, and Dignum 2005], but none of them has been officially adopted by FIPA.

1.3 Service orientation

In the years 2000s technologies such as Web services [Booth et al. 2004] and Grid computing [Kesselman and Foster 2004] emerged and matured, with the support of both the research community and the industry, with a more pragmatic perspective with respect to multi-agent systems.

These technologies are based on the concept of service-orientation [Erl 2004]: a distributed system is comprised of units of service-oriented processing logic (the *services*) which hide their internal logic from the outside world and minimize dependencies among them. Recently some of these service-oriented technologies are converging into a single overarching framework, called Service Oriented Architectures (SOA). Such framework created a collection of best practices principles, the main ones agreed by the SOA community being:

- *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 minimises dependencies and only requires that they maintain an awareess 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 minimise 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.

Service-orientation 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.

Analogously, 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 were many service-oriented architectures, frameworks and initiatives made available. The OASIS Service Oriented Architecture Reference Model (SOA-RM) [MacKenzie et al. 2006] was 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) [Booth et al. 2004] identified 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 [Condry, Gall, and Delisle 1999], Grid [Kesselman and Foster 2004] and JINI [Arnold 1999] architectures were all service-oriented architectures, proving that SOA was widely accepted as a paradigm.

The standard technology chosen by the community for the implementation of SOA systems based on these architectures and frameworks was the Simple Object Access Protocol (SOAP) [Curbera et al. 2002], a specification for exposing action-based method signatures.

Thanks to the closeness between agent-oriented and service-oriented approaches, there was some cross-fertilization between both technologies. The SOA community already identified some potential to integrate agent research in SOA, i.e., Paurobally et. al proposed to adapt and to refine Multiagent Systems research community results to facilitate the dynamic and adaptive negotiation between semantic Web Services [Paurobally, Tamma, and Wooldridge 2005]. Foster, Jennings and Kesselman already identified in [Foster, Jennings, and Kesselman 2004] the opportunity to have some joint research between the Grid and Agents communities.

The approach of service-orientation for distributed computing was more pragmatic than Multi-agent systems, mainly due to its less formal theoretical background and the fact that it was thought more as a set of principles rather than a set of languages, protocols and behavioural requirements. However, the large amount of architecture specifications aforementioned failed due to their adding a large amount of complexity in design and implementation, and the expectatives were not met [Manes 2009]. Also, technologies that were embraced by the SOA community, such as SOAP and XML, were quickly replaced by less complex alternatives like Representational State Transfer (REST), which specified how to expose resources rather than actions, and JSON, and for a few years both the acronym 'SOA' and the service-orientation principles have been apparently discarded.

With the growth of the REST popularity, there has been in the last years a wide adoption of this technology in the form of Application Programming Interfaces (APIs). The main motivation behind this – fairly rapid – switch in paradigms was [Pautasso, Zimmermann, and Leymann 2008] the lower complexity, in the short term, of specifying schema-less resources (REST) instead of actions, along with their complex types (SOAP). Also, REST services are coreographed by the developer in a hardcoded way, instead of being forced to follow an OASIS or W3C specification which was proven to be a cumbersome procedure [Michlmayr et al. 2007].

At the same time that the service-orientation principles lost popularity, the REST API community embraced the Agile principles, i.e., rapid prototyping, continuously changing requirements, less prioritisation of processes [Bloomberg 2013]. However, this did not stop the appearance over time of the same questions that motivated the service-orientation principles: how to deal with effective decoupling of concerns, how to use declarative specifications to coordinate and orchestrate services, how to deal with state, how to achieve governance, etc.

The community has recently coined a term to refer to *"service-orientation done right"* without explicity using the term SOA: *Microservice Architecture*. Although the term has been used in the Internet for a while, Martin Fowler defined the term in the seminal [Fowler and Lewis 2014] with an already mature definition based on the success use cases of Amazon and Netflix. According to this article, *"an application is a suite of small services (microservices), each running in its own process and communicating with lightweight mecanisms, often an HTTP resource API"*.

We argue that both terms (Microservice Architecture and SOA) are equivalent, because the definition of service-orientation, and the frameworks and W3C-OASIS proposals built around it, are two separate things, regardless of how much they have been confused by the public. For this reason, in the rest of this document we will keep using the term SOA, but it is important to note that we aim to encompass both the old and the new adopted meanings with it. After all, a service as an abstract concept is still the same, independently of whether it exposes actions or resources: it is an agent, a means of communicating with other agents and provide information or trigger external actions or events, and the same (conceptual) solutions apply to both concepts with respect to governance.

1.4 Summary of this chapter

In this chapter we have described several paradigms that are well suited to materialise the concept of *computation as interaction* in the sense of computing as a social activity resulting from the joint actions of human and/or software agents.

First of all, we have discussed the concept of agency and its suitability to describe individuals in a distributed systems. From there, we have seen two types of systems of such agents: 1) Multi-agent Systems, strongly founded on academic formalisms, and 2) Service-Oriented Architectures, more industry-oriented and therefore more pragmatic.

As we will see in Chapter 2, this divergence affects how social reality can be captured and processed by each type of agent. More pragmatism means more flexibility, but that is not necessarily a good thing; while academic foundations already provide concepts closer to social or institutional theories. Our aim is to combine both worlds with computational artifacts both flexible and formally sound.

CHAPTER 2

Norms

Omar: 'I mean, I do some dirt, too, but I ain't never put my gun on nobody that wasn't in the game.'
Bunk: 'A man must have a code.'
Omar: 'Oh, no doubt.'

The Wire
David Simon

Agency and normativity are two very closely related concepts, and it is often said that there cannot exist one without the other [Korsgaard 2008]: if an agent wills an end, it is committing oneself to realising that end, and therefore willing an end is a first-personal *normative* act. Because agents are capable of assigning normative principles to themselves, they can also *accept* new ends to will, which carry this normative weight.

When we have groups of agents instead of individuals, the ends of the agents make interaction complex: different agents may will the same end or, as is often the case, the ends of two agents may be conflictive. Human agents have been capable of translating first-personal normativity to forms of collective normativity: what we call *norms*.

In literature, the concept of *norms* has been defined from several perspectives [Vázquez-Salceda 2003]: as a rule or standard of behaviour shared by members of a social group, as an authoritative rule or standard by which something is judged, 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.

What is common to each of these meanings of the concept is that they provide some sort of guidelines in the interaction between agents. Norms are therefore the foundational element for shaping *societies* of agents, from human organisations to mixed human/artificial socio-technical systems.

In this chapter we will summarise the relevant work on norms from two different perspectives. First of all, we will see how norms can be categorised, and in the next section we will explore different ways in which they can be implemented.

2.1 Types of norms

There is no consensus on how to categorise norms, although it is accepted that norms, as an abstract concept, influence the behaviour of rational agents. This influence can depend on how the norms are promulgated, enforced, or whether the norms explicitly relate to an end or otherwise the norm is created as an end by itself. With respect to these issues, [Elster 1989] makes a distinction between three types of norms:

- Social norms: they are not outcome oriented and tend to be enforced by individual feelings towards the other agents, and thus not necessarily benefit the participants, e.g. *it is customary to salute people when arriving to the office*.
- Moral norms: they are consequentialist, i.e. an act is morally right if and only if that act maximises the global *good*, where good is a moral concept rather than an economic or functional optimal, e.g. *you should not commit adultery*.
- Legal norms: enforced by specialists out of self-interest – it is their job, e.g. *it is obliged to drive on the right lane of the road whenever possible*.

Social norms (also called conventions, private norms or informal norms) are an interesting phenomenon: because they are not outcome oriented, they are not entirely rational from an individual perspective. They are generally unconditional, and in those cases when they are not, they are not future-oriented (due to the lack of a goal). In fact, in order to become social, a norm must be shared and sustained by approval and disapproval, and therefore a social norm is an end in itself.

It might seem from this definition that social norms cannot be the foundation of a society, but they are indeed. They allow the developing of mechanisms of reciprocity and cooperation [Bicchieri 2006], because their acceptance gives shape to groups of agents distinctive to others and allows collective behaviour patterns to emerge in those gaps where the other types of norm – moral, legal – cannot reach.

Moral and legal norms share the fact that they are outcome-oriented, and therefore following them or not requires individual rationality. The difference between both, according to [Elster 1989], is the motivation of the enforcer: legal norms are enforced by people who would lose their job otherwise. However, this distinction is not entirely clear [Posner 2009] – for example, does the Bible promulgate legal or moral norms? After all, the Catholic Church is an institution that has enforced such norms for many years and we could argue that such enforcement involved self-interested agents in many cases. On the other hand, moral norms require moral agents: consequentialism can only be understood and accepted by agents that can reason about the ends from an individual perspective. This is not the case with artificial systems (at least, for now), because these have usually predefined or pre-programmed goals.

This very much seems clear: in both cases, enforcing norms requires the use of language in order to communicate them due to the fact that they deal with ontological concepts that are not necessarily physical. This is a distinction with respect to social norms, as it is proven that they can appear merely by means of observation between similar agents [Bicchieri 2006].

As formulated in Research Question 2, our objective is to produce computational elements that enable individual elements in distributed systems as first-class members of socio-technical systems. In this context, it seems fit to understand that the norms affected by such systems are closer to what [Elster 1989] categorises as legal norms: outcome-oriented towards an external goal, *written* in some kind of language and enforced by specialists. In order to simplify, from now on we will simply refer to this type as *norm*.

Independently of this distinction of norms based on their outcome and enforcement, [Searle 2009] argues, following Speech Act Theory, that norms (called *rules* in this book), due to their performative value, are used by agents to effectively create *social reality*, that is, socially accepted facts that are ontologically subjective (their existence depends on the mental states of a rational agent) but epistemically objective (they are agreed upon by a collective of agents).

As mechanisms to create this social reality, Searle identifies two types of norms (called *rules* in this book):

- Constitutive rules: they declare institutional (social) facts from brute facts – which are ontologically objective – or from other institutional facts.
- Regulative rules: they declare constraints on the behaviour of agents with respect to their relationship with institutional facts.

A constitutive rule typically has the form:

X Counts as Y in C

where X is a brute or institutional fact, Y is the institutional fact being created, and C is a normative context. An example of constitutive rule in action is two people considered as married (institutional fact) because they were declared as such by a person who had the power to do so. An example of regulative rule related to this institutional fact is for example that the married couple are in the mutual obligation of declaring taxes together (in some countries and/or regions).

In summary, regulative rules can be seen as describing ideal situations from an institutional perspective in terms of obligations, prohibitions and permissions; whereas constitutive rules allow to construct social reality by expliciting the relationship between brute facts and institutional events. The main difference between both types of norms is that while constitutive rules, by their very nature, are categorical, regulative rules are conditional, in the sense that they specify every applicable condition of each particular norm[Boella and Torre 2004].

In some works it has been proposed to consider a third kind of norm – *regimented norms* [Torre et al. 2004]. These are considered hard constraints as opposed to regulative and constitutive, which are soft constraints. In this context, a hard constraint cannot be avoided, i.e. the norm is enforced by imposing a restrictive environment, whereas soft contraints can be violated, i.e. not accepting a declaration or not fulfilling an obligation.

In distributed systems, regimented norms already exist in the form of implementation decisions that directly or indirectly create those contraints (by design). We are more interested on the aspects of soft constraints, especially in distributed systems where the internal specifications of services are usually hidden from other parties.

2.2 Representing norms

In the previous section we have explored the concept of norm from a theoretical point of view, i.e. what kinds of norms we can encounter. A different topic, however, is how to give shape to norms. In this section, we describe the most relevant ways to describe norms in governed distributed systems[1] in the chronological order in which they were proposed.

2.2.1 Deontic logic

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* [Lokhorst 1999], based on the *classical propositional calculus*. Von Wright [Wright 1951] 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:

[1]This discards some mechanisms, such as game theory, that are more suited for theoretical or simulated use cases.

[2]Some 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.

$O(p \to q) \to (O(p) \to O(q))$	(KD1 or K-axiom)
$O(p) \to P(p)$	(KD2 or D-axiom)
$P(p) \equiv \neg O(\neg p)$	(KD3)
$F(p) \equiv \neg P(p)$	(KD4)
$p, p \to q \vdash q$	(KD5 or Modus Ponens)
$p \vdash O(p)$	(KD6 or O-necessitation)

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.

In fact, Deontic Logic is not a logic of norms, but a logic of propositions stating the existence of norms [McNamara and Prakken 1999]. However, this makes them suitable for implementing regulative rules as defined in Section 2.1. Although there have been recent attempts to make regulative rules concrete by the reduction to constitutive rules [Aldewereld et al. 2010b], in general regulative norms based on deontic statements have been the most common way to represent normative constraints in multi-agent systems. In such systems, thus, norms are expressed as computer-readable specifications based on deontic logics.

However, operationalisation of such norms is not straightforward. As already mentioned, deontic statements express the existence of norms, rather than the consequences of following (or not following) them [Walter 1996]. In order to implement agents and institutional frameworks capable of reasoning about norms, we need to complement deontic logics with semantics defining fulfillment and violation – among other operational normative concepts.

Standard Deontic Logic or *KD* [Meyer and Wieringa 1991] 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 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 [Kanger and

Stenlund 1974; Lindahl 2001; Pörn 1974], and allows to express direct and successful operations: E_iA 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 [Santos and Carmo 1996; Santos, Jones, and Carmo 1997]. Thus, the modal operator G allows to express indirect but successful operations: G_iA 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_iA means that an agent i *attempts to make it the case* that A.

Examples of work on deontic logic to materialise norms are abundant and for many different purposes, i.e., compliance [Alvarez-Napagao et al. 2011; Cardoso and Oliveira 2009; Criado et al. 2010; García-Camino et al. 2006; Governatori and Rotolo 2010; Oren et al. 2009], verification [Ågotnes, Hoek, and Wooldridge 2010; Koo 2008; Lomuscio, Qu, and Raimondi 2009; Prisacariu and Schneider 2009], or agent behaviour [Aldewereld et al. 2005; Kollingbaum 2005; López, Luck, and d'Inverno 2004; Meneguzzi and Luck 2009; Panagiotidi and Vázquez-Salceda 2011]. While it is true that most of them define semantics to interpret norms, there seems to be a disconnection between such semantics and either 1) the deontic logics they are supposed to be based upon; or 2) the operational level closer to the actual practical implementation. For instance, [Governatori and Rotolo 2010] defines a norm-operationalization language that can be connected with higher level abstractions, but it is not clear whether it can be translated into generic rule-based languages. [García-Camino et al. 2006] presents a rule-based language with constraints, with an implementation on Prolog, on top of which other higher-level languages can be formalised, but with no direct relationship to deontic logics. On this line of work, approaches such as [López, Luck, and d'Inverno 2004; Oren et al. 2009] define clear operational semantics by the use of syntax loosely inspired by, but not directly related to, deontic statements.

In [Aldewereld 2007], Aldewereld extends the ideas of [Vázquez-Salceda 2004] by applying parts of the methodology to highly-regulated environments (environments governed by lots of complex norms). The author formalises the implementation of norms from an institutional point of view. He defines the abstract norm, and extends this formalism to the norm frame needed for the representation of concrete norms. More specifically he uses an linear-time temporal logic, based on temporal logic which he extends with deontic operators to express the obligations, permissions and prohibitions that are in the norms. He then introduces the operational constraints by which they implement the norm frame to obtain the normative institutions. The author then proceeds to specify (formal) methods for the implementation of norm enforcement and the (automatic) creation of protocols (based on constraints specified by the norms).

What is somehow missing in general in the literature is a clear separation between what an abstract norm and a particular (contextual) instantiation of the norm. This problem was already discussed by Abrahams and Bacon in [Abrahams and Bacon 2002]: *"since propositions about norms are derived from the norms themselves, invalid or misleading in-*

ferences will result if we deal merely with the propositions rather than with the identified norms[3] *that make those propositions true or false"*. This issue is not banal, as it has implications on the operational level: in order to properly check norm compliance, norm instantiations have to be tracked in an individual manner, case by case.

We find useful, at this point, to stress the fact that the lifecycles of a norm, and of a norm instance, should be differentiated because they are different in essence. The lifecycle of a norm (see Figure 6.3) deals with its validity in the normative system, while the lifecycle of a norm instance (see Figure 2.2) deals with the fulfillment/violation of the particular instance.

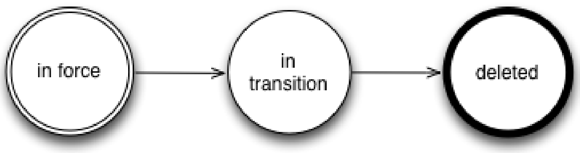


Figure 2.1: *Norm lifecycle*

❧

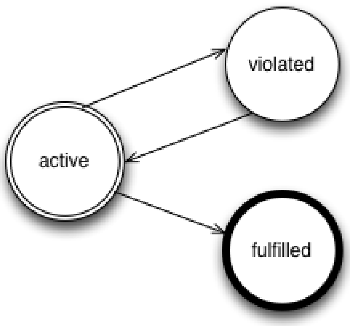


Figure 2.2: *Norm instance lifecycle*

❧

Abrahams and Bacon [Abrahams and Bacon 2002] solve this problem by means of *occurrences* of the predicates contained in the deontic operator, but there are cases in which this can be insufficient, e.g., when the obligation defines a deadline or its instantiation depends on contextual information. More recently, some works have been advancing in the direction of tackling this issue. For example, by treating instantiated deontic statements as first-class objects of a rule-based language [Cardoso and Oliveira

[3]From now on, we will denote such identified norms as *norm instances*.

2009; Governatori 2005]. However, as these deontic statements are already implicitly identifying the norm instance, there is no explicit tracking. Other approaches declare the norm only at the abstract level and the tracking of the norm instance, and implicitly of the norm instance lifecycle, is purely done at the operational level [Alvarez-Napagao et al. 2011; Criado et al. 2010; Modgil et al. 2015; Oren et al. 2009].

In summary, there are many approaches that tackle different parts of the formalization of norm operationalization. One of the purposes of our work will be, thus, to complement these approaches by filling the gaps that exist between the deontic statements and other suitable more simpler operationalisations, with special focus on norm instances.

Therefore, while it is true that deontic logics provide a level of expressivity that is ideal, being somehow close to what is used in human language for norms, its complexity has proven difficult to deal with in practical systems. These limitations may impact directly or indirectly on important factors in distributed systems, esp. regarding real-time performance and scalability. We keep exploring options based on simpler formalisms in the following subsections.

2.2.2 Production systems

Event-Condition-Action (ECA) rules were first defined in the context of database management systems [McCarthy et al. 1989] and are the simplest form of rule possible. Such rules are first-class objects of a language, with clear semantics: when an event occurs, the condition of each rule is evaluated; if the condition is satisfied for a particular rule, its action is executed. Therefore, they have the same expressivity and complexity of a typical *if-then-else* branching in a programming language, but targetting a virtually infinite stream of input events. Because of their simplicity, they have been applied in almost every domain of practical applications.

Many libraries and frameworks implement mechanisms for efficient execution of ECA rules systems. These are commonly referred to as *rule engines* or *production systems*. In this thesis we will use as a reference the semantics for production systems proposed in [Cirstea et al. 2008], summarised in the following paragraphs.

Considering a set $\mathcal{P}$ of predicate symbols, and an infinite set of variables $\mathcal{X}$, where a fact is a ground term, $f \in \mathcal{T}(\mathcal{P})$, and $\mathcal{WM}$ is the *working memory*, a set of facts, a production rule is denoted `if` p, c `remove` r `add` a, or

$$p,\ c \Rightarrow r,\ a,$$

consisting of the following components:

- A set of positive or negative patterns $p = p^+ \cup p^-$ where a pattern is a term $p_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and a negated pattern is denoted $\neg p_i$. p^- is the set of all negated patterns and p^+ is the set of the remaining patterns
- A proposition c whose set of free variables is a subset of the pattern variables: $Var(c) \subseteq Var(p)$.

- A set r of terms whose instances could be intuitively considered as intended to be removed from the working memory when the rule is fired, $r = \{r_i\}_{i \in I_r}$, where $Var(r) \subseteq Var(p^+)$.
- A set a of terms whose instances could be intuitively considered as intended to be added to the working memory when the rule is fired, $a = \{a_i\}_{i \in I_a}$, where $Var(a) \subseteq Var(p)$.

Definition 1 *A set of positive patterns* p^+ matches *to a set of facts* $\mathcal{S}$ *and a substitution* σ *iff* $\forall p \in p^+, \exists t \in \mathcal{S}, \sigma(p) = t$. *Similarly, a set of negative patterns* p^- dismatches *a set of facts* $\mathcal{S}$ *iff* $\forall \neg p \in p^-, \forall t \in \mathcal{S}, \forall \sigma, \sigma(p) \neq t$.

A production rule $p \Rightarrow r, a$ *is* $(\sigma, \mathcal{WM}')$-fireable *on a working memory* $\mathcal{WM}$ *when* p^+ *matches with* $\mathcal{WM}'$ *and* p^- *dismatches with* $\mathcal{WM}$, *where* $\mathcal{WM}'$ *is a minimal subset of* $\mathcal{WM}$, *and* $\mathcal{T} \models \sigma(c)$. □

Definition 2 *The* application *of a* $(\sigma, \mathcal{WM}')$*-fireable rule on a working memory* $\mathcal{WM}$ *leads to the new working memory* $\mathcal{WM}'' = (\mathcal{WM} - \sigma(r)) \cup \sigma(a)$. □

Definition 3 *A general production system* $\mathcal{PS}$ *is defined as* $\mathcal{PS} = \langle \mathcal{P}, \mathcal{WM}_0, \mathcal{R} \rangle$, *where* $\mathcal{R}$ *is a set of production rules over* $\mathcal{H} = \langle \mathcal{P}, \mathcal{X} \rangle$. □

These semantics allow us to treat production systems, which lack a high level of expressivity, as a theoretical framework in top of which we can reduce formalisms that involve more complex rules.

2.2.3 Service-level agreements

The concrete approaches used for enforcement in SOA are based on Service-Level Agreements (SLA). A 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 providers' commitments, a lot of specification work was 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 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 [Ludwig, Dan, and Kearney 2004], 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) [Ludwig et al. 2003] is a framework targeted at defining and monitoring SLAs for Web Services [Keller and Ludwig 2003]. 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 [Andrieux et al. 2005], 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.[4] WS-Agreement is quite often used with the conversation definition language WSCL [Banerji et al. 2002]. 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 [Bougiouklis 2008] was 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 combined centralised 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 was an attempt to connect, at a high level, Service Level Agreements and Multi-Agent Systems, focusing on semi-automated ne-

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

gotiation and offline monitoring of contracts, and including interesting features such as matchmaking, negotiation, and Virtual Organisation (VO) evaluation. The main issue concerning PANDA for its use in a generic contracting architecture was that its main focus was in the domain of ERP-solutions and not domain-independent.

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) [Paschke 2005]. However, RBSLA does not explicitly adopt the Web service-technology. Standard generic rule and inference engines such as Mandarax [Dietrich 2005], based on RuleML [Boley 2006], and Prova [Kozlenkov and Schroeder 2004] 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).

2.3 Summary to this chapter

In this chapter, we have defined norms as proper mechanisms for describing behavioural constraints in the interactions between agents, as agency and normativity are intimately related concepts. Furthermore, we have identified types of norms depending on whether they are social, moral or legal, on one hand, and whether they are used to specify a vocabulary (constitutive) or to define rules based on that vocabulary (regulative). The combination of constitutive and regulative norms effectively allows us, by definition, to design systems that have social reality as a first-class citizen.

Additionally, we have seen several forms of norm definition, at different levels of expressivity and originally designed to fulfill disparate objectives: 1) Deontic Logic, very close to the expressivity of human norms, 2) Production Systems, simple rules matching conditions to produce actions, and 3) Service-Level Agreements, in the middle of the former two with regards to complexity and expressivity, and created with service-orientation as a target.

Because deontic logics are usually too complex to be seamlessly translated into the implementation level of practical systems, and production systems are not expressive enough to capture the complexity of constraints usually written by humans, service-level agreement was at its time a fair attempt at dealing with this issue. However, as seen in Section 1.3, SOA-related proposals were gradually abandoned in favour of more *ad hoc* solutions, i.e. hardcoding orchestration and constraints in service implementation. For example, Amazon and Netflix enforce an approach based on adding the constraints directly into the implementation of the services, making developers follow the contracts agreed upon by the consumer and the provider instead of delegating these tasks to the service itself [Fowler and Lewis 2014].

This means that there is still an opportunity of providing a solution that tries to tackle governance of distributed systems for SOA in a more flexible and adaptive way than merely by design. However, at the same time this carries the challenge of overcoming the problems of past attempts, which proved too complex, and delivering a solution that is, at the same time, efficient, expressive, and easy to adopt.

CHAPTER 3

Normative systems and governance

WOMAN: 'Well, how'd you become king then?'
KING ARTHUR: 'The Lady of the Lake, her arm clad in the purest shimmering samite held aloft Excalibur from the bosom of the water, signifying by Divine Providence that I, Arthur, was to carry Excalibur. That is why I am your king.'
DENNIS: 'Listen, strange women lyin' in ponds distributin' swords is no basis for a system of government. Supreme executive power derives from a mandate from the masses, not from some farcical aquatic ceremony.'

Monty Python and the Holy Grail
TERRY GILLIAM & TERRY JONES

Until now, we have seen what is considered as an agent and how artificial agents can form distributed systems (Chapter 1), and how we can apply the concept of normativity to them (Chapter 2). Norms can, therefore, be used to shape and guide the behaviour of individuals in distributed systems – in *societies*. However, in order to achieve this, there has to be a framework, conceptually apart from the mere topology of the distributed system and the implementation of the individual agents, that allows agents to understand the norms and reason about them. In other words, agents have to share a *normative context*.

In research, the systems, usually based on social abstractions that help building and enabling such normative contexts are called *Normative Systems*. The main idea of Normative Systems is that the interactions among a group of (software) agents are ruled by a set of explicit norms expressed in a computational language representation that agents can interpret. Although some authors only see norms as inflexible restrictions to agent behaviour, others see norms not as a negative, constraining factor but as an aid that guides the agents' choices and reduces the complexity of the environment, making the behaviour of other agents more predictable.

There are two main approaches in the literature about how to design such Normative Systems. On one hand, Normative Systems can be defined in terms of institutions, a concept that can be derived from norms at a theoretical level. Such systems are usually heavily grounded on existing formalisms, especially from Institutional Theory, Sociology and Deontic Logics. On the other hand, we can find more pragmatic approaches that have been evolving from the requirements detected from practical systems, and are grounded on sets of best practices created from the bottom (actual services) up.

In this chapter we will explore both sides of the same problem, tackled from each perspective.

3.1 Institutional theory applied

For Searle, the existence of institutional facts pressuposes the existence of institutions and an institution (or society) is constituted by means of constitutive rules: the structure of an institution is, in fact, the logical structure of its counts-as rules [Searle 2005]. Therefore, *"an institution is any system of constitutive rules of the form* X counts as Y in C *"*. Once an institution is established by such rules, it serves as a structure on top of which institutional facts (and therefore social reality) can be created.

Language is essential for the existence of institutions. For example, we can have language without money, but not money without language. Physically, a bill is a piece of paper. This is the only *ontologically objective* fact we can infer from inspecting it. In order to enable agents to understand that the bill is, in fact, money, there must be a medium of representation, a means to represent the fact that the bill is money. In the case of human institution, this medium is human language (in the broad sense, including non-verbal communication or symbolism). For example, it would be hard to understand how two agents could play a game of chess without a proper verbal or written representation of its (constitutive and regulative) rules.

Types of institutional facts can be simple declarations, such as *declaring marriage*. However, as seen in Section 2.1, institutional facts can carry obligations in the form of regulative rules. Thus, institutions not only create a terminology for the society, but they also enable *deontic powers* [Searle 2009]: rights, duties, obligations, authorisations, permissions, empowerments, requirements, and certifications. Such deontic powers allow societies to create institutional facts of utmost importance for the interaction between agents, e.g. private property, government, contracts, friendship, family, money.

This gives us an account for a formal definition of institution, providing us with a combination of constitutive and regulative rules that give shape to the interaction between agents in terms of deontic powers. Therefore, institutions define normative contexts.

Institutions, seen as norm providers and enforcers, can be used to solve the following issues in the context of distributed 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.

Attempts of producing theoretical frameworks that convert and extend the concept of institution into frameworks (normative systems) have been produced by [North, Nørgaard, and Swedberg 1990; Richard Scott 1998, 2001], in an area of study called Institutional Theory. These authors defined ways to relate institutional norms with both agents and the environmental context of the agents, and to analyse such relationships in terms of (mainly economical) performance.

For [Richard Scott 1998], institutions are open systems where the environment or context has an important influence as it *constraints*, *shapes*, *penetrates* and *renews* the society or organisation. In this 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 specifications of (proper/unproper) behaviour to be followed by individuals.

Scott [Richard Scott 2001] defines a framework to describe the interactions between institutions and organisations. 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.

On the other hand, [North, Nørgaard, and Swedberg 1990] argues that 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.

In this work, North studied the effect of institutions, in the sense of normative systems, on the behavior of human organisations, 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.

Therefore, the creation of institutions provides trust among parties even when they do not have much information about each other. 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 [Ostrom 1986] 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 organisation as being an instance of a particular kind of institution, and then use this knowledge to predict other norms that could be applicable.

The various attempts at translating these theoretical frameworks to the context of distributed systems have received many names: *Socio-technical Systems*, *Norm-governed Systems*, or *Electronic Institutions*. What all of them have in common is that they intend to model of human institutions through the specification of their 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.

3.1.0.1 State of the art on normative frameworks

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 applying the institutional abstraction to distributed systems.

Organisational approaches have been quite successful in applying model social order to software agent societies. For example, V. Dignum *et al.*, in OperA, ALIVE, and OperA+ [Dignum, Weigand, and Xu 2002; Dignum et al. 2008; Jiang, Dignum, and Tan 2012] integrated and evolved some previous work in order to present a framework composed by three interrelated models:

- The *organisational model*, which decribes the structure of the organisation 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 organisation, the framework defines *interaction structures*. These structures define patters 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. Constitutive and regulative norms, based on the OperettA meta-model [Aldewereld and Dignum 2010a], are executed and monitored in parallel to detect violations without compromising agent autonomy[1].

Artikis, Pitt *et. al* [Artikis, Sergot, and Pitt 2009] use the concept of Norm-governed Computational Societies to refer to a concept analogous to institutions. Organisational specifications are computed by means of Event Calculus [Farrell et al. 2005] combined with the action language [Sergot 2010]. As first-class objects, these specifications allow declaring powers, sanctions, obligations, prohibitions and permissions. However, when dealing with services it might be desirable to specify constraints in terms of events or states instead of actions, and this framework does not allow that.

JaCaMo [Boissier et al. 2013] is a development platform in which the paradigms of agent orientation, organisation orientation and environment orientation are put together by joining, among others, JASON [Bordini and Hübner 2006], with the organisational framework with institutional capabilities MOISE [Hannoun et al. 2000]. The concept of organisational artifact is used in JaCaMo to define regimented or regulative norms that can be monitored in run-time. However, the operationalisation of norms is coupled to the platform, and thus cannot be easily applied to services that do not belong to it.

Magentix [Alberola et al. 2008] is a Multi-agent system with a wide range of coordination capabilities: virtual organisations, BDI reasoning and argumentation. Their definition of institution is slightly different: institutions are virtual organisations with

[1]In Chapter 6 we describe the mechanisms used to effectively monitor norms modelled in the OperettA meta-model.

a predefined library or interaction protocols allowed. Their norm language is suitable for regulative (not constitutive) rules, and although there are plans to integrate the norms as purely regulative, at the moment they are only used as regimented.

The SMART agent architecture presented by d'Inverno and Luck [d'Inverno and Luck 2004] is based on an agent specification framework developed on top of the Z specification language [Spivey 1989]. This framework defines concepts such as *objects*, *agents* and *autonomous agents*[2]. This framework was extended by López y López, Luck and d'Inverno [López and Luck 2002; López, Luck, and d'Inverno 2001] 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 [Esteva, Padget, and Sierra 2002; Noriega 1997; Rodríguez-Aguilar 2003], later evolved to EIDE [Esteva et al. 2008], is a proposal that treats electronic institutions as a type of *dialogical system* where all the interactions inside the institution are a composition of multiple dialogic activities, that is, message exchanges. These interactions, called *illocutions* [Noriega 1997], 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 [Cortés and Rodríguez-Aguilar 2000; Garcia et al. 1999; Matos and Sierra 1999]. Still, ISLANDER/EIDE presents some drawbacks in its application to complex domains such as distributed systems, 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 (performative structures, scenes, transitions), thus reducing the autonomy of the agents.

InstAL [De Vos, Padget, and Satoh 2010] is a proposal for the specification of executable institutions by means of logic programming (Answer Set Programming). First-class elements include fluents, events, obligations, and consequence relations. The specification of institutions in InstAL allow for constitutive rules, thus enabling the generation of institutional events such as violation events. As the specification is instantiated by the framework, the agents are virtual elements that are created by InstAL at runtime. Also, the operationalisation is based on the same language (ASP), and therefore there is no known way to reduce the semantics to external services in order to externalise processes such as monitoring or enforcement.

[2] *Agents* are defined as objects with goals, while *autonomous agents* are defined as objects with motivations.

Some approaches have been designed specifically for the *e*-commerce and *e*-business fields, such as SMACE [Cardoso and Oliveira 2000] or the Contractual Agent Societies (CAS) [Dellarocas 2000]. 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*-organisation may need to define in order to ensure an appropriate behavior of the society of agents.

An evolution of SMACE is ANTE [Oliveira et al. 2014], a framework that improves over the former by adding a terminology layer, including vocabulary such as obligation, violation, deadline, violation or fulfillment. The concept of institution is materialised in ANTE as a collection of services available for contracting, and contracts dynamically determine what an organisation is. However, in ANTE external entities are agentified into the system, that is, services must be wrapped in order to participate in the environment through the ANTE platform.

3.2 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, an organisation, or a group a people*. This can be done by constituting laws, rules, standards, or principles [Simpson and Weiner 2003]. Governance, in practice, is defined by Finkelstein [Finkelstein 1995] 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 [Rosenau and Czempiel 1992] 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 organisations 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.

Corporate governance, for example, 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 [Monks and Minow 2004]. The actors involved in corporate governance are the shareholders, the board of directors, the management and other stakeholders[3], although in most cases it is a delegated responsibility on the managers and the board, bounded by public legislations and regulations.

[3]Customers, employees, suppliers or the community at large are examples of stakeholders.

For Colley et al. [Colley et al. 2003], 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 organisation defines and fulfills its obligations to its constituencies or stakeholders in view of conflicting interests?
- *Effectiveness:* how does the board ensure that they and their 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?

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 [Simonsson and Johnson 2006] 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 organisation.

In most of the organisations, 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. [Webb, Pollard, and Ridley 2006], 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 generated by organisational IT investments [Weill 2004]. 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 [Trites 2004].

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 [webMethods 2006]. 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.

One main concern about SOA is that is supposes a paradigm shift for the industry [Ibrhaim et al. 2007]. 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*.

3.2.1 A definition of SOA governance

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.

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 identify how the community is doing at heading in this direction and which companies are on the good track and what they lack [Fulton, Heffner, and D'Silva 2008; Kenney and Plummer 2008].

SOA Governance[4] is an emergent concept in the SOA community used for activities related to exercising control over services[Linthicum 2008]. It is a form of electronic governance that has its focus on distributed services and composite architectures, more concretely on SOA scenarios.

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

The business SOA community is already aware of the importance of successfully defining and promoting a standard which may help the stakeholders in using SOA Governance and acquire its benefits.

3.2.2 Institutional governance based on agents

There are three steps that define SOA Governance management [webMethods 2006]. Design-Time Governance deals with the definition and application of policies that will govern the design and implementation of Web services in the organisation, prior to

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

their deployment in the actual business environment. During Run-Time Governance, policies are defined and enforced in order to govern the deployment, execution, and use of the Web services. Eventually, Web services are supposed to be redesigned and reimplemented in order to adapt to business evolving requirements. Change-Time Governance focuses on how the changes on the services affect the behavior of a whole SOA environment.

There are three steps that define SOA Governance management [webMethods 2006]:

- Design-Time Governance: definition and application of policies that will govern the definition and implementation of Web services in the organisation, prior to their deployment in the actual business environment.
- Run-Time Governance: definition and enforcement of policies that will govern the deployment, execution, and use of the Web services.
- Change-Time Governance: Web services are supposed to be redesigned and reimplemented in order to adapt to business evolving requirements. This type of governance focuses on how the changes on the services affect the behavior of a whole SOA environment.

The approach currently used in SOA Governance management is based on adding additional Web services in the SOA environment. The main components are: a registry (a central catalog for business services), a repository (a database of governance policies and metadata), policy enforcement points (services responsible for the enactment of the policies), a rule engine (automatic system that manages the enforcement of the policies), and a configuration environment (user interface for the configuration and definition of policies and governance workflows).

The company *webMethods* (a subsidiary of Software AG) defined in a white paper [webMethods 2006] 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 summarised in Figure 3.1 and described in the following sections.

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 organisational architecture processes.

These include:

- Establishing organisational 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.

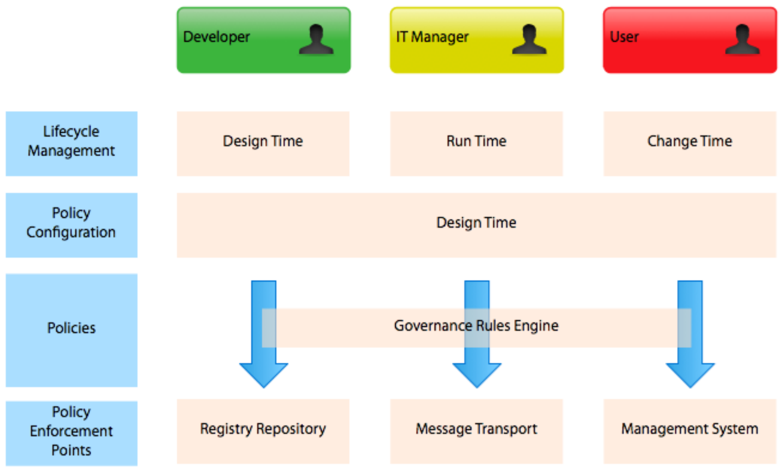


Figure 3.1: *Key components of a SOA governance system*

❧

- 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 thoroughful evolution of the architecture over time. The majority of organisational 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.

3.2.2.1 *Service lifecycle governance*

A fundamental point in SOA [Erl 2004] 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 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 categorise 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 organisational 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 [Nezhad et al. 2006]: usage guidelines that ensure Web services implemented on different platforms are interoperable, by automatically verifying service schemas, validating namespaces, and other controls.

If an organisation 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 organisational standards for business function encapsulation, reusability, reliability, and so on. By ensuring that these reviews are performed by appropriate members of the organisation, 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 organisational requirements.
- Ensuring the strategic design of services and validating that their interfaces and implementation conform to established design patterns and other organisational 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 authorised 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 authorised consumers possessing the appropriate permissions, and that data is encrypted if required.
- Validating that services operate in compliance with prescribed organisational 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 actions. 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 organisational goals. This need arises from the fact that services exist to support organisational functions as well as the inter-organisational relationships and dependencies that are implicit in SOA, particularly when services are exposed and invoked across organisational boundaries. Since changes are generally initiated and driven by the requirements, users need to be participants in the governance lifecycle.

3.2.3 Technologies for SOA governance

If we follow the requirements structure, a SOA governance system should facilitate service-level governance across the lifecycle from design-time to run-time to change-time. It should allow polices 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.

3.2.3.1 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 [Clement 2005] 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 centralised catalog of services is significant from an organisational 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 categorised and organised.
- 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 [Bellwood 2001] 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.

3.2.3.2 *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 organisational 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 centralised repository is particularly important for codifying and enforcing a single *official* set of policies across the organisation. 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-organisational boundaries.

3.2.3.3 *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 enforcement[5]. 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.
- *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.

[5]The 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 [webMethods 2006].

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 [Schmidt et al. 2005]. 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 unauthorised 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 optimise 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.

3.2.3.4 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 [Menzel et al. 2007], 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 organisation. A more robust model, which is the basis for policy federation, is to enable both centralised 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.

3.2.3.5 *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 authorised 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, organisations might define approval and notification workflows, exception alerts, and a variety of other process steps.

3.3 Summary of this chapter

In this chapter, we have seen how the problem of the need for control in distributed has been tackled from two different research perspectives. The first one, taking theoretical frameworks from Economics (Institutional Theory) and Philosophy as foundations, in order to provide logically sound and formally comparable solutions. The second one, created as sets of requirements in a pragmatic way from actual experience.

An immediate conclusion we can extract is that they are two completely different abstractions, but that if we are to combine high-level coordination mechanisms and distributed systems there are compromises to be made. Our contribution will be to leverage both points of view by applying the institutional abstraction to practical systems in a way that can be reduced to governance architectures.

To do so, we will need a proper language to specify norms, based on the proposals studied in Chapter 2, to allow individuals in distributed systems to understand, reason about, and communicate about high-level constraints. We will also need to create transformations of such language to operational semantics that reduce the complexity of their interpretation. Finally, we will need to make this reduction computable.

PART II

PRACTICAL REDUCTIONS

CHAPTER 4

The challenge

The bottom line is that simplicity is a choice. It's your fault if you don't have a simple system.

Simple Made Easy
Rich Hickey

As seen in previous chapters, monitoring is one of the important points of abstraction for *e*-institutions to be deployed and integrated in SOA. *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 [Sommerville 2006]. 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 interacting computational components, it has been shown that this definition of monitoring is not enough to deal with common problems in distributed scenarios [Groth, Luck, and Moreau 2004; Matyska et al. 2007].

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 occur at different points in time and thus are apparently not related to each other, but are in fact parts of a complex transaction.

4.1 Solving the drawbacks of Service-Level Agreements

We have pointed out in Section 2.2.3 the flaws that the Service-Level Agreements (SLA) in SOA initiatives have, especially related to the lack of expressiveness allowed: in general, the available SLA approaches focus on monitoring simple metrics rather than enabling high levels of expressitivity. Although in some cases there is the possibility of adding declarative abstractions in the form of simple rules, there is still a lack of solutions that offer the declaration of constraints via modal or deontic expressions.

With the increasing growth in the complexity of tasks and responsibilities of distributed systems as seen in Chapter 1 and especially in Section 1.3, specifying interaction contraints is also progressively more difficult to manage. Additionally, these constraints are (still) a product of stakeholders – designers, developers, users –, rather than being produced automatically by computers. The numerous attempts, as summarised in Section 2.2.3, to raise expressivity and therefore cope with this complexity are proof that there is a need to bring the design of such constraints closer to the way humans think of, write and understand norms. This is not possible to accomplish with simple rules [Grangard et al. 2001].

The drawbacks described 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 3.1, 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.

In our previous work, one notable attempt to fill this gap was by using Provenance architectures[1] [Vázquez-Salceda and Alvarez-Napagao 2009], which in the context of IT allow capturing causal relationships between events produced by actors of distributed systems, with annotated semantics, while providing a framework for reasoning about complex distributed processes as a structured graph. Therefore, rather than focusing on storing interactions and message exchanges in a free format, using provenance allowed us to not only store internal states of actors but also the relationship between these interactions, and the temporal or causal relationship between states with interactions or other states. Provenance handling effectively enables interpretable monitoring of loosely-coupled distributed complex processes, and we took advantage of this in order to build a real-time verification system for compliance of complex rules translated from regulations and protocols. However, provenance-based systems require both a deep knowledge of the domain and the use of an event model tighly coupled to the Open Provenance Specification [Moreau et al. 2008]. In this thesis we focus on achieving the same level of compliance checking and expressivity whilst achieving greater flexibility, and as analysed in the state of the art (see Section 3.1), normative systems are a candidate to provide both.

In open societies, where heterogeneous agents are self-governed autonomous entities pursuing their own goals, it is a challenge to provide mechanisms to ensure that the system as a whole behaves as expected without undermining agents' autonomy. An approach widely explored in literature is to add a social layer describing and governing

[1]Provenance architectures allow storing and querying provenance, where the provenance of a data item is represented in a computer system by a set of assertions made by the actors involved in the process that created it.

the actors' expected behaviour in order to produce desirable – and avoid undesirable – situations. In such models, goals and interactions are usually not specified in terms of the mental states of individual agents, but in terms of social or organisational concepts such as roles – function, position –, groups – communities –, norms – regulations, policies – and communication protocols – including ontologies. In these cases, agents are seen as actors that perform the role(s) described by the social/organisational design. In short, Normative Systems are a socially-inspired approach for the governance of distributed, agent-oriented systems, where the expected behaviour of agents is described by means of an explicit specification of norms.

As such, there are many works (see Section Section 3.1) in literature that explore the use of normative specifications to define the acceptable behaviour of a set of agents – including services – in a given environment. Norms such as regulative rules provide a way to introduce flexibility in the specification of desired actor behaviour in a shared context, as they typically specify the boundaries of what is acceptable, but not how exactly to behave. However, regulative rules do not entail any value of truth but instead express the mere existence of a norm. Norms have indeed been studied from multiple perspectives, but while formalizations tend to be disconnected from possible implementations due to the lack of differentiation between abstract norm and norm instantiation, on the other hand implementations tend to be weak groundings of deontic logics, tightly coupled to one particular implementation domain. In this dissertation, we build upon previous work to fill these gaps by reducing from deontic statements to structural operational semantics, with intermediate steps in linear temporal logics. Finally we hint at the feasibility of the translation of these semantics to actual implementation languages in different domains.

4.2 Filling the gaps on normative systems

There is a lot of work on normative systems' formalization (mainly focused in Deontic-like formalisms [Wright 1951]) which is declarative in nature, focused on the expressiveness of the norms [Dignum et al. 2004], the definition of formal semantics [Aldewereld et al. 2005; Boella and Torre 2004; García-Camino et al. 2006; Oren et al. 2009] and the verification of consistency of a given set [Governatori and Rotolo 2010; Lomuscio, Qu, and Raimondi 2009].

There are some works that focus on norm compliance and norm monitoring [Ågotnes, Hoek, and Wooldridge 2010; Alvarez-Napagao et al. 2011; Criado et al. 2010; García-Camino et al. 2006; Governatori and Rotolo 2010; Oren et al. 2009] with varying degrees of covered abstraction level and allowed flexibility. Also there is some work on how agents might take norms into account when reasoning [Aldewereld et al. 2005; Kollingbaum 2005; López, Luck, and d'Inverno 2004; Meneguzzi and Luck 2009; Panagiotidi and Vázquez-Salceda 2011], but few practical implementations exist that cover the full BDI cycle, as many approaches do not include the *means-ends reasoning step* (that is, dedicing HOW to achieve WHAT the agent is aiming for).

Most of these approaches focus on regulative norms rather than on substantive norms, and lack a proper implementation of the ontological connection between brute events and institutional facts.

Therefore, it is not uncommon in some scenarios, i.e., when building a norm-aware multi-agent system, to implement normative reasoning from several points of view at the same time. For instance, in a system with institutional-level norm enforcement, there is normative reasoning at – a minimum of – two levels: the institutional compliance and the individual agent reasoning. In such scenarios, the following are relevant issues that should be taken into consideration:

1. There is a need to formally connect the deontic aspects of norms with their operationalisation, preserving the former.
2. From a practical point of view, abstract norms have to be distinguished from their actual instantiations. For each abstract norm, many instantiatons may happen during the norm's lifetime.
3. Ideally, the operational semantics should be formalised in a way that ensures flexibility in their translation to actual implementations while ensuring unambiguous interpretations of the norms. For instance, the semantics used by a society compliance mechanism, and the semantics integrated in the reasoning cycle of its individual agents, must be aligned to avoid, e.g., the agent norm reasoning mechanism stating that no norm has been violated while the compliance mechanism states that the agent has violated some norm instance.

Some works in the literature present solutions that tackle these issues separately (e.g. recent works such as [Meneguzzi et al. 2015] tackle 1. and 2.), but it is hard to find a proposal that manages to fill the gap left by these three issues at the same time. In this document, we present a proposal that combines ideas from many pieces of work, coming from different formalizations while using [Oren et al. 2009] semantics as a foundation, in order to achieve a deontic-based norm definition with a direct reduction to temporal logics which, in turn, can be translated into rule-based operational semantics.

4.3 Our proposal

We present a computable language, a formalism for the monitoring of both regulative (deontic) and substantive (constitutive) norms based on Structural Operational Semantics, its reduction to Production Systems semantics and our current implementation compliant to these semantics.

To ensure that the formalism used by the normative monitor is also compatible with an agents' practical reasoning, the work presented in this document is built upon the combination of work previously done separately in [Alvarez-Napagao et al. 2011; Panagiotidi and Vázquez-Salceda 2011] to have a unified normative framework with two different implementation perspectives: planning and monitoring.

The rest of this Part II contains the materialisation of this proposal. In Chapter 5, we present our language of norms for services and agents. We then present the operational semantics for this language in Chapter 6, and finally we present our normative monitor, along with the reduction to production systems and a implementation of it, in Chapter 7.

CHAPTER 5

Towards a lightweight language for norms

‘Mind you, I don’t object to foreigners speaking a foreign language. I just wish they’d all speak the same foreign language.’

Avanti!
BILLY WILDER

As mentioned in Section 2.1, in order to interpret part of their social reality, agents in a socio-technical system need ways to be able to track the status of all the social norms that may apply to them and to the behaviours of other actors that may affect them (directly or indirectly). From a computational perspective, the first foundational element needed for an effective and efficient monitoring of high-level norms in distributed systems – both agent-oriented and service-oriented systems – is a proper language to describe such norms in a declarative way. The following is a summary of the requirements that such a language such have:

- The language should be compact and lightweight. Norms expressed in the language should have the minimum number of elements possible, as they are going to affect the size of the payload in the communications inside a distributed system.
- The language should have a formal grounding, linked to the literature on the topic of normativity. The research on this field has implications on a broad range of areas of knowledge: logics, philosophy, sociology, economics. A language that tackles directly the concept of norm should allow any derived work to be put into context from a formal perspective in order to be used or applied by the related areas of knowledge.

- The language should incorporate not only the concept of regulative norm but also the concept of constitutive rules (counts-as rules, see Section 2.1). Not only the behavioural constraints themselves are relevant, but also the ontological transformation rules from brute facts to institutional events.
- The language should enable dealing with meta-concepts from the domain of normative contexts, e.g. norm, obligation, prohibition, etc. This would enable reasoning at the level of the norms, beyond the domain and the predicates used in the constraints themselves.
- The language should allow for an efficient operationalisation. That is, any derived execution framework for the detection of normative states based on the language should be computable in efficient time/space.

Some SLA languages deal with the first requirement, but in general none of the other requirements are tackled by them. In this Chapter we present a language that fulfills these requirements. In order to do so, we take an incremental approach, showing a first language we built especifically for contracts in SOA without proper counts-as rules, trying to tackle directly the issues of SLAs; and the extensions done afterwards for adding constitutive rules and generalising the approach for distributed systems in general and multi-agent systems in particular.

5.1 A language for contract representation

A contract [Simpson and Weiner 2003] 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 has already been explored by Dellarocas with the concept of Contractual Agent Societies [Dellarocas 2000].

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 monitoring 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 took place in these two directions, in the scope of the IST-CONTRACT project[1], where we propose a move to a more flexible contracting mechanism for Service Oriented Systems based on the following three main elements:

[1]The IST-CONTRACT Project was a project (FP6-034418) funded under the 6th Framework Programme of the European Commission. The Contracting Language presented in Section 5.1 was developed during the project lifetime.

- 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.

This section presents 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 Section 5.1.1 we describe the conceptual layers of our contracting language. Section 5.1.2 analyses the important elements of the contract representation model. Section 5.1.3 explains how messaging and communication are achieved between the contracting parties.

As an example, we will adopt one of the use cases produced during the IST-CONTRACT, consisting in a web-service system enabling agreements between several parties (insurance companies, repair companies, and a third kind of party called *Damage Secure* which acts as a broker between the former) in a car insurance scenario. In this scenario, insurance companies file repair requests and *Damage Secure*, as an intermediary, is responsible for finding the best possible repair company for the job[2].

5.1.1 Layers/Elements of the Contracting Language

This contracting language[3] aims to define the way agents can exchange information during contract establishment, contract execution and contract termination. During these phases, agents may exchange not only information about the contracts themselves, but also about any related event, predicate or action that may be related to such contracts. Therefore it needs to define the communication structures required to flexibly exchange information about parties, processes, world states, ontological definitions and so on.

[2]For a more detailed description of this use case, see [Panagiotidi et al. 2008].

[3]My contribution to the work presented in this section was twofold: leading the analysis of the state of the art on existing contracting languages, identifying the required layers and elements, and a strong and continuous involvement in the design process of the contracting layers. These contributions were made along with Sofia Panagiotidi, Roberto Confalonieri, Javier Vázquez-Salceda and Steven Willmott.

The contracting language separates concerns between different layers of communication:

1. The *Domain Ontology Layer*, containing the domain ontology: terms, predicates and actions (e.g. *car*, *workshop*, *repair*), to unify definitions used between parties and avoid ambiguity or misunderstanding.
2. The *Contract Layer*, defining constraints in the form of clauses: deontic statements about the parties' obligations, permissions and prohibitions. The basic elements of such statements are any predicate or action defined in the Domain Ontology (e.g. the workshop is *obliged to* repair the car in 2 days).
3. The *Message Content Layer*, defining what can be contained inside messages communicated between parties. This includes meta-predicates at a normative level, such as statements about contracts (e.g. *active/inactive*, *fulfilled*, *complete/incomplete*), actions about contracts (e.g *accept*, *sign*, *cancel* a contract), as well as statements and actions about concepts from the Domain Ontology.
4. The *Message Layer*, describing how agents can model their performative attitudes towards the contents of the messages (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[Agents 2002a]) are included as part of the message envelope.
5. The *Interaction Protocol Layer*, allowing sequences of messages between agents to be specified by means of contracting protocols. These protocols structure interaction by defining sets of acceptable sequences of messages which would fulfil the goal state of the protocol (e.g. agreeing on contract termination).
6. The *Context Layer*, representing the interaction context where contractual parties will carry out the obligations, permissions and prohibitions agreed in the contract.

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* are central elements in the framework, as they semantically connect references to the same entities and objects made at different layers. These ontologies also ensure that the same definition for actions and predicates is consistently used, avoiding interpretation errors. This last aspect is specially relevant in contracts, as all parties should have the same understanding about the agreed terms in the contract. 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*.

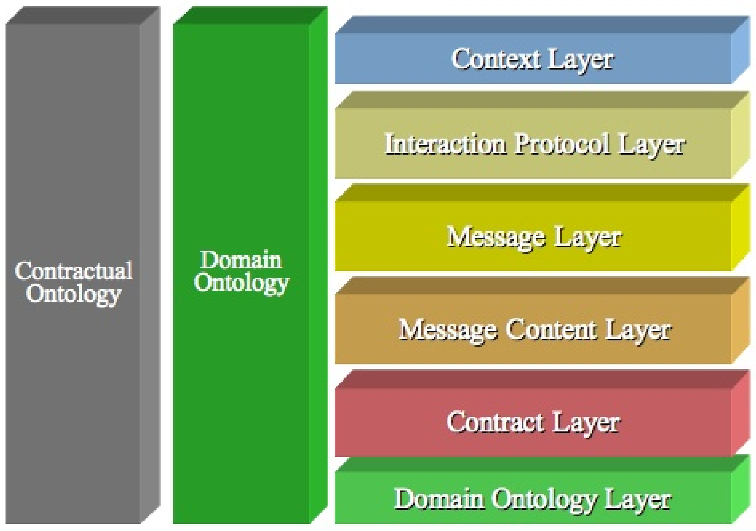


Figure 5.1: *General view of the Contracting Framework*

❧

- 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*.

5.1.2 Contract representation

The Contract Data Model is an XML based representation suitable to represent both fully defined contracts or contract templates (partially defined contracts)[4]. It is composed by a name, starting and ending dates and three main parts: contextualization, definitions and clauses. Figure 5.3 depicts this structure, following the WC3 standard XML Schema Definition. A summary of the semantics of the visual elements used in the diagrams can be found at Figure 5.2.

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

[4]My contribution to the work presented in this section was twofold: leading the research work on the state of the art the language was built upon and a strong and continuous involvement in the design process of the contracting language. These contributions were made along with Sofia Panagiotidi, Roberto Confalonieri, Javier Vázquez-Salceda, Steven Willmott and Sandra Ortega-Martorell.

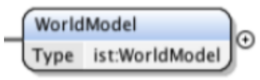
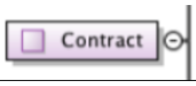
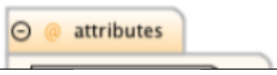
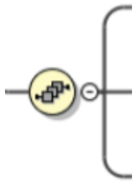
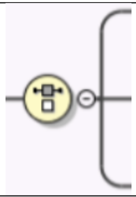
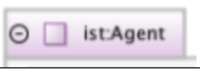
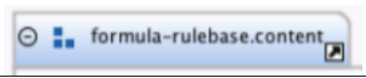
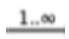
Icon	*Concept*
	Dependency (another element)
	Root of the element
	List of attributes
	Sequence of elements (e.g. children)
	Enumeration (exclusive choice)
	Imported namespace
	Imported XML Schema
	Cardinality

Figure 5.2: *Visual elements for the XML Schema-based diagrams*

❧

allowed by leaving the ending date blank). The rest of the elements are described in the following sections.

The contract can be contextualised by defining (see Figure 5.4): 1) a list of references to parent contracts, the interaction contexts of which will be the parents of the context created for this contract; and/or 2) a reference to the contract template, if there is any.

5.1.2.1 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 (see Figure 5.5).

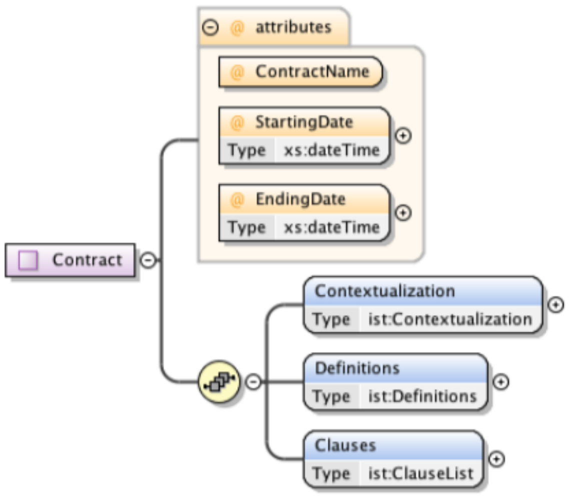


Figure 5.3: *Root elements of the Contract representation*

❧

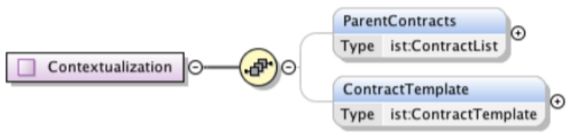


Figure 5.4: *Contextualisation element in the Contract Language*

❧

5.1.2.2 *Parties*

The *contract parties* are 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 5.6).

Following is an excerpt of the contract document that models the parties:

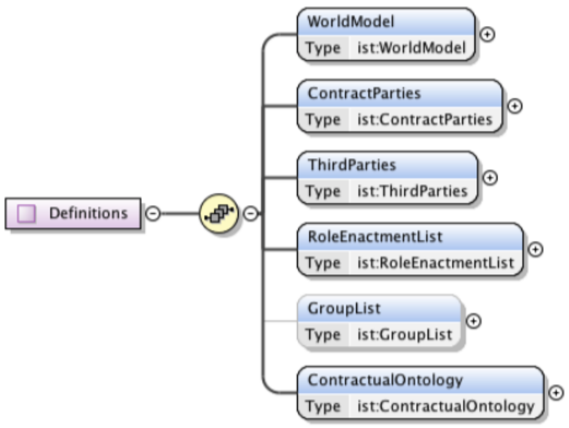


Figure 5.5: *Contractual definitions element in the Contract Language*

❧

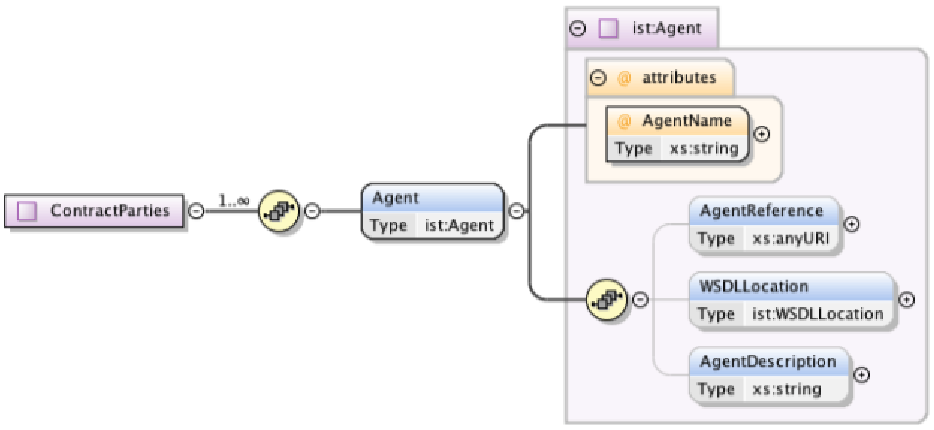


Figure 5.6: *Parties element in the Contract Language*

❧

```
<ContractParties>
<Agent AgentName="Feel Safe">
  <AgentReference>feelsafe.com:8080/FS</AgentReference>
  <AgentDescription>Car Insurances</AgentDescription>
 </Agent>
 <Agent AgentName="Damage Secure">
  < AgentReference>damsec.org:8080/DS</AgentReference>
  <AgentDescription>Service Broker</AgentDescription>
 </Agent>
 <Agent AgentName="Fast Fix">
  <AgentReference>fastfix.com:8080/BR</AgentReference>
  <AgentDescription>Bob's Garage</AgentDescription>
 </Agent>
</ContractParties>
```

5.1.2.3 *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 the definition of responsibilities from the specific agents that will have to fulfill them. This decouplement 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 5.7 shows the structure of a Role-enactment list.

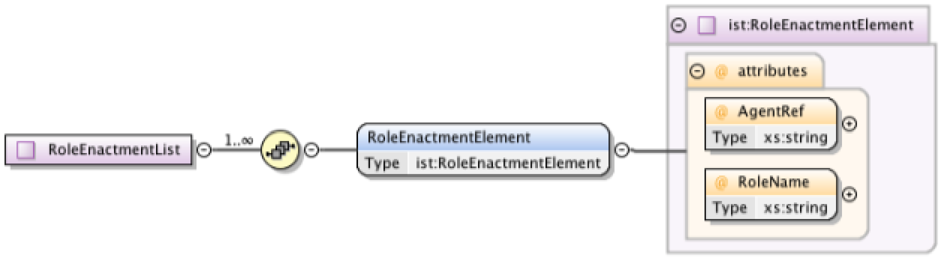


Figure 5.7: *Role enactment list element in the Contract Language*

❧

For example:

```
<RoleEnactmentList>
<RoleEnactmentElement
    AgentName="Feel Safe" RoleName="Insurance Company"/>
  <RoleEnactmentElement
    AgentName="Damage Secure" RoleName="Broker"/>
  <RoleEnactmentElement
    AgentName="Fast Fix"   RoleName="Repair Company"  />
</RoleEnactmentList>
```

5.1.2.4 *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 5.8.

5.1.2.5 *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

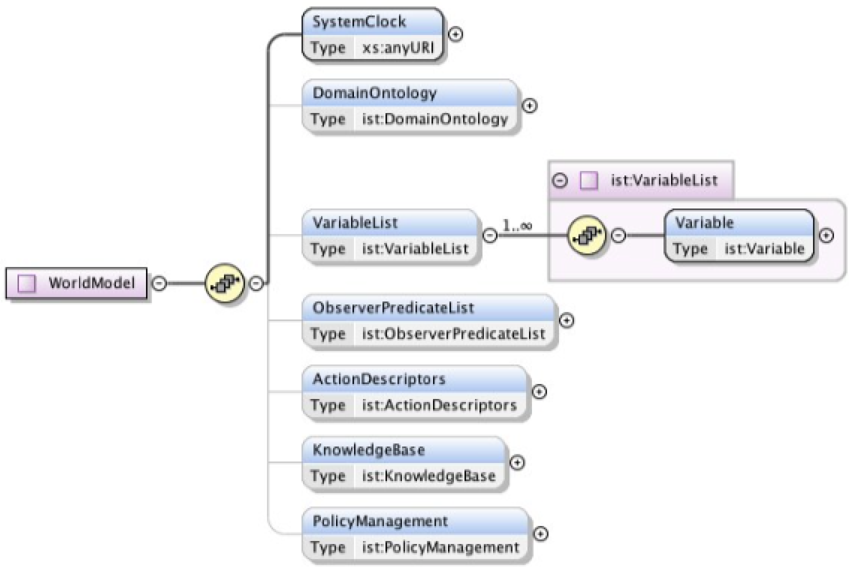


Figure 5.8: *Representation of the world model element in the Contract Language*

❧

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[Vázquez-Salceda 2004].

5.1.2.6 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 [Aldewereld 2007], 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* [Vázquez-Salceda, Aldewereld, and Dignum 2004].

A clause (Figure 5.9) 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 (a clause specifying the deontic consequences of violating another clause).
- *Exploration (or maintainance) 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.

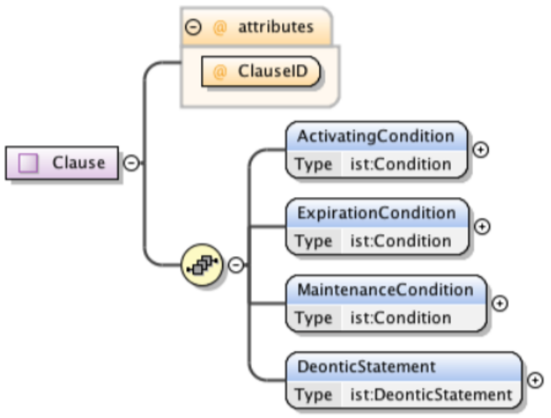


Figure 5.9: *Representation of the clause element in the Contract Language*

❧

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 (see Figure 5.10):

- *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 (see Figure 5.11). This object can be an action or a state. If it is an action, this action will have to be executed in order

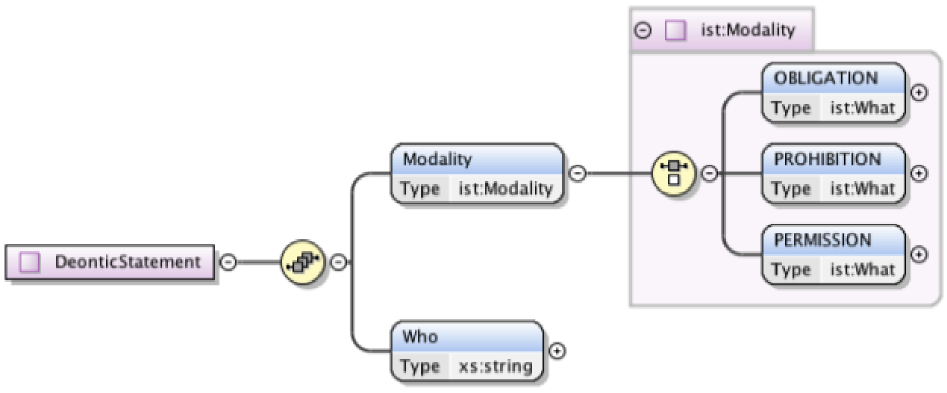


Figure 5.10: *Representation of the deontic statement element in the Contract Language*

❧

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.

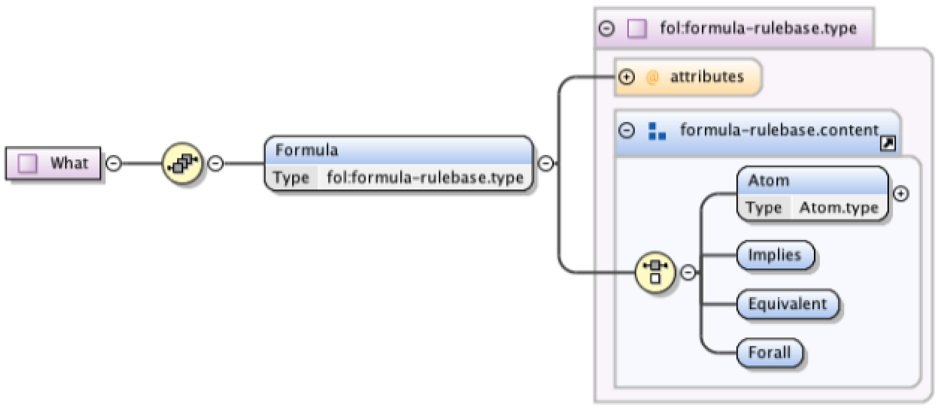


Figure 5.11: *Representation of the What element in a clause in the Contract Language*

❧

If the Modality is Obligation, the parties responsible will have to execute/ensure the object of the norm in order to fulfil the clause. If it is a Prohibition, the parties will have to avoid executing/ensuring the object, and if it is a Permission, the parties are informed that they can execute/ensure it.

Clauses can be used in two ways:

- 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 the insurance company before April 10 if the report of the repair is ready". Such clause, which in dyadic deontic logic would be formalised as:

$$\begin{aligned}&exists(RepairReport(car)) \rightarrow \\ &O_{RepairCompany}(sendRepairCompleted(car, DamageSecure, RepairCompany)\\ &\qquad\qquad \leq April\_10)\end{aligned}$$

And 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>
    <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 this case that the obligation should be met before a given deadline. The *DeonticStatement* specifies that this is an obligation related to one agent, R1,[5] enacting the *Repair Company* role, and it consists of one action (sending the repair report).

5.1.2.7 Contractual Ontology (IST-Contract)

As mentioned in Section 5.1.1, 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

Table 5.1 and Table 5.2 show the pre-defined actions and predicates created to express statements about contracts are shown. 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.

5.1.2.8 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.

[5]It 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.

action	*attributes*	*precond.*	*postcond.*

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))

end	P: Instance of Party
	C: Instance of Contract
		exists(C)
		initiated(C)
			happened(terminate(P, C))
			$\neg$ initiated(C)
			active(C) $\rightarrow$ $\neg$ active(C)
			terminated(C)

withdraw	P: Instance of Party
	C: Instance of Contract
		exists(C)
		initiated(C)
		committed(P, C)
			$\neg$ committed(P, C)
			$\neg$ initiated(C)

request_cancel	P: Instance of Party
	C: Instance of Contract
		exists(C)
		initiated(C)
			happened(request_cancel(P, C))

create	P: Instance of Party
	C: Instance of Contract
		$\neg$(exists(C))	exists(C)
			happened(P, create(P, C))

cancel	P: Instance of Party
	C: Instance of Contract
		exists(C)
		initiated(C)
			active(C) $\rightarrow$ $\neg$ active(C)
			$\neg$ initiated(C)
			happened(cancel(P, C))
			cancelled(C)
			terminated(C)
			$\forall$ PT : committed(PT, C)
			$\neg$ committed(PT, C)

get-contract	P: Instance of Party	exists(C)	happened(get-contract(P, C))

update	P: Instance of Party
	PT: Instance of Party
	C: Instance of Contract (old version)
	NC: Instance of Contract (new version)
		exists(OC)
		OC is valid
		NC is valid
			exists(NC)
			initiated(OC) $\rightarrow$ (initiated(NC)
			$\wedge\neg$ initiated(OC))
			active(OC) $\rightarrow$ (active(NC)
			$\forall$ PT : committed(PT, OC)
			committed(PT, NC)
			$\forall$ PT : committed(PT, OC)
			$\neg$ committed(PT, OC)
			happened(update(P, OC, NC))

Table 5.1: *Library of contract-related actions*

predicate	*description*	*attributes(terms)*
happened(A)	An action happened at some point in the past	A: Instance of Action
before(T)	The current time is before some time	T: Instance of Time
after(T)	The current time is after some time	T: Instance of Time
committed(P, C)	A party is committed to a contract	P: Instance of Party C: Instance of Contract
all-committed(LP, C)	All parties have committed to a contract	LP: List of Instances of Party C: Instance of Contract
violated(P, C, LO)	A party has violated obligations of a contract	P: Instance of Party C: Instance of Contract LO: Instances of Obligations
obliged(P, C, O)	A party is bound to an obligation within a contract	P: Instance of Party C: Instance of Contract O: Instance of Obligation
ended(C, T)	A contract has been ended due to a violation	C: Instance of Contract T: Instance of Time
initiated(C)	A contract has been signed	C: Instance of Contract
active(C)	A contract is taking action and has not yet terminated	C: Instance of Contract
cancelled(C)	A contract has been cancelled	C: Instance of Contract
fulfilled(P, C, O)	A party has fulfilled an obligation within a contract	P: Instance of Party C: Instance of Contract O: Instance of Obligation

Table 5.2: *Library of contract-related predicates*

❧

As mentioned in Section 5.1.1, 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 and stable during the contract lifetime, e.g. the names for the known insurance companies could be added to the ontology, as they are quite stable through time and rarely appear or dissappear.

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 5.1.3.1). 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 5.1.2).

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.

5.1.3 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 [Agents 2002b] 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[6].

5.1.3.1 *Message Structure*

FIPA [Agents 2000] 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 [Agents 2002b], where the message body contains the usual FIPA proposed attributes, i.e. sender, receiver, performative, language, content, etc.

5.1.3.2 *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.

[6]My contribution to the work presented in this section was twofold: leading the research work on the state of the art that was used as our foundations and the design of 7 of the 14 interaction protocols. These contributions were made along with Sofia Panagiotidi, Roberto Confalonieri, Michal Jakob, Javier Vázquez-Salceda and Steven Willmott.

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 5.1.2.7 and 5.1.2.8 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 [Agents 2000]. 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 [Finin et al. 1994], allowing for better constructs with better semantics and lower complexity.

A subset of FIPA-SL, namely FIPA-SL2, adapted to an RDF representation[7], 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 (*forall*, *exists*) and reference operators (*iota*, *any*, *all*), which are needed in order to give the expressivity needed for flexible contract-related communication.

5.1.3.3 Performatives

The full set of FIPA-ACL [Agents 2002b] *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 [Tuomela 1996]:

- **suggest**: The action of submitting a suggestion for the sender and the receiver agents to perform a certain action.
  <i, suggest (j, <i, act>, <j, act>)>
  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.
  <i, consent-suggestion (j, <i, act>, <j, act>)>
  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.
  <i, dismiss-suggestion (j, <i, act>, <j, act>, ψ)>
  Agent i informs j that, because of proposition ψ, i does not have the intention for i and j to perform action act.

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

action	*protocol(s)*
end	Contract Fulfilment Protocol With Notary
request_cancel	Contract Violation Protocol Without Manager
	Contract Violation Protocol With Manager
create	Simple Contract Creation Protocol With Notary
	Simple Contract Creation Protocol Without Notary
	Sideways Contract Creation Protocol With Notary
	Sideways Contract Creation Protocol Without Notary
	Simple Negotiation Creation Protocol With Notary
	All Parties Agree Creation Protocol Without Notary
	All Parties Agree Creation Protocol With Notary
cancel	Contract Cancelling by Agreement Protocol Without Notary
	Contract Cancelling by Agreement Protocol With Notary
update	Contract Modification/Update Protocol With Notary

Table 5.3: *Library of contract-related actions*

❧

5.1.3.4 Protocols

Contract execution requires agent behaviors which should be well defined and designed, in order for the contract lifecycle to be successfully 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 a significant part of every agent's expected behavior.

The IST-CONTRACT project contributed a library of protocols which allow agents to communicate whenever one of the phases is taking place are defined. In some cases, more than one protocol is provided, since there can be more than one way in which a stage can successfully take place (i.e. when a contract is being created, there might or might not be a negotiation phase involved) and the users should be able to pick the most appropriate for the case.

In Table 5.3 we show the correspondence between the protocols and the contract-related actions listed in Table 5.1.

As an example of how these protocols work, Figure 5.12 and Figure 5.13 depict examples of protocols expressing, respectively, the creation of a contract between two agents without an intermediary and the modification of a contract with the presence of a notary.

The Sideways Contract Creation Protocol Without Notary consists of several interactions between two contract parties (Initiator and Participant):

1. The Initiator sends to the Participant a SUGGEST message, communicating the

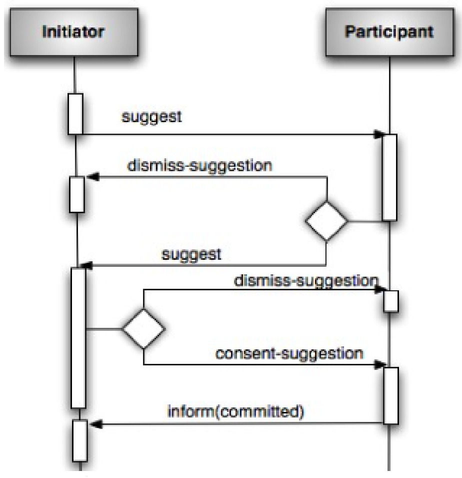


Figure 5.12: *Sideways Contract Creation Protocol Without Notary*

❧

Participant a proposal of the terms of the contract for his part

2. The Participant replies:
    a) with a DISMISS-SUGGEST message if the former does not accept the terms of the contract part proposed or,
    b) with a SUGGEST message of his part of the contract terms filled
3. The Initiator replies:
    a) with a DISMISS-SUGGESTION message if the former does not accept the proposed contract terms and the protocol finishes with no agreement and with no contract creation
    b) with a CONSENT-SUGGESTION message if the former agrees on the terms of the new

The Contract Modification/Update Protocol With Notary consists of the following interactions between two contract parties (Initiator and Participant) and the notary:

1. The Initiator sends to the Participant a SUGGEST message, communicating the Participant a proposal of the terms of the modified/updated contract for his part
2. The Participant replies:
    a) with a DISMISS-SUGGEST message if the former does not accept the modified/updated terms of the contract and the protocol finishes with no agreement and with no contract modification/update, or,
    b) with a CONSENT-SUGGESTION message if the former agrees on the terms of the modified/updated contract

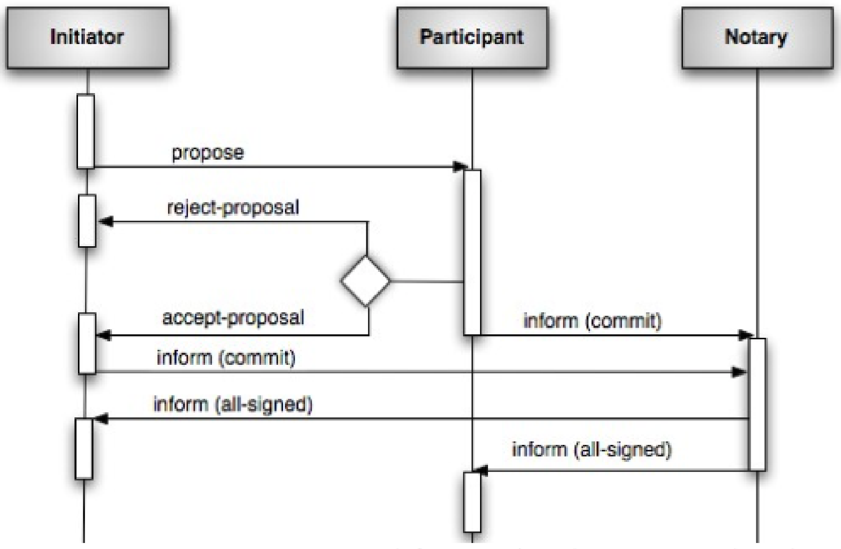


Figure 5.13: *Contract Modification/Update Protocol With Notary*

❧

3. Initiator and Participant both send a INFORM message to the notary informing him they have committed to the modified/updated contract
4. the notary replies to the Initiator and Participant with an INFORM message, confirming the mutual agreement of the modified/updated contract

5.1.4 A proposal for operational semantics

This section introduces a summary of the first proposal for operational semantics based on the language described in Section 5.1[8]. It is important to note that this is not a contribution made by us but by other members of the IST-CONTRACT, and it is included in this section as it is relevant enough to at least give some insight on it because it will be relevant to follow how the language evolved and what is different on the operational semantics we will introduce in Chapter 6. Therefore, these semantics are only briefly described in the section, while their full motivation and formalisation can be found at [Oren et al. 2009].

The semantics presented here operationalise norms by tracking the status of each of them in terms of their lifecycle: norms are *abstract* until they are *instantiated*, and each instantiated norm can be in one of four states: simply instantiated, expired, holding true, or holding false.

Assuming the existence of a theory Γ allowing for the interpretation of the status of norms, that includes a normative environment theory Γ_{NEnv}:

[8]The work presented in this section was carried mainly by Sofia Panagiotidi, Nir Oren, Sanjay Modgil, Javier Vázquez-Salceda and Michael Luck. However, I had a minor contribution on it as I attended and participated in the process of formalisation of the semantics as part of my IST-CONTRACT duties.

Definition 4 *An abstract norm is a tuple of the form*
$\langle Modality, ActivatingCondition, ExplorationCondition,$
$EndCondition, Who\rangle$
where all the elements of the tuple correspond to the concepts described in Section 5.1.2.6 □

$ActivatingCondition$ corresponds to a *wff* ϕ_{AC} which, when entailed by Γ, must be entailed as the fully grounded ϕ'_{AC} in order that the abstract norm can be instantiated and thus become active. We call this substitution of variables S such that $\phi'_{AC} = S(\phi_{AC})$.

Definition 5 *An abstract norm is instantiated by* Γ*, obtaining an instantiated norm:* $\langle Modality, ActivatingCondition', ExplorationCondition',$
$EndCondition', Who'\rangle$
where:

- $\Gamma \vdash ActivatingCondition'$*, where* $ActivatingCondition'$ *is fully grounded such that* $ActivatingCondition' = S(ActivatingCondition)$
- $ExplorationCondition' = S(NormCondition)$
- $EndCondition' = S(EndCondition)$
- $Who' = \{X | \Gamma \cup \{ActivatingCondition'\} \cup \{S(Who)\} \vdash X\}$

□

At this point, we can define a series of predicates that will allow us tracking the normative state of the instantiations:

Definition 6 *Given an instantiated norm* in *such that*
$in = \langle Modality, ActivatingCondition, ExplorationCondition, EndCondition, Who\rangle$
Then, for $N \in ExplorationCondition, EndCondition$*:*
$\Gamma \vdash N' \rightarrow holds(in, N) = true$
where N' *is entailed with all variables in* N *grounded; otherwise* $holds(in, N)$ *evaluates to* $false$*.* □

The formal definition of normative state identifies those instantiated norms whose exploration condition evaluates to true, those whose exploration condition evaluates to false, and those whose end condition evaluates to true:

Definition 7 *Given a set of instantiated norms* INS*, a normative state* NS *is a tuple of the form:*
$\langle NSTrue, NSFalse, NSExpires\rangle$
where:

- $NSTrue = \{in \in INS | holds(in, ExplorationCondition) = true\}$
- $NSFalse = \{in \in INS | holds(in, ExplorationCondition) = false\}$
- $NSExpires = \{in \in INST | holds(in, EndCondition) = true\}$

□

Temporality is reflected in these semantics by assigning an initial normative state NS_0 and virtually infinite normative states NS_i where $i > 0$. With these definitions as a starting point, the operational semantics define a series of predicates that allow to detect normative state changes for a given instantiated norm. For example:

Definition 8 $\Gamma \vdash instantiated(\lceil NS_i \rceil, in)$ *iff*
$inst\_norms(NS_i) \wedge (in \notin inst\_norms(NS_{i-1}) \vee in \notin NSExpires_{i-1}$.
We define by default $\Gamma \not\vdash instantiated(\lceil NS_0 \rceil, in)$. □

The rest of predicates can be extracted from $\langle NSTrue, NSFalse, NSExpires \rangle$ in a similar manner, by tracking changes between two normative states NS_i and NS_{i+1}. These predicates include *active()*, *expires()*, *violated()* or *fulfilled()*.

These operational semantics provide a correct formal grounding of the language presented in Section 5.1. However, they are lacking some features needed for practical reasoning in monitoring (as well as in enforcement and individual normative planning). For example, there is no explicit normative life-cycle for norms and norm instances defined, and the one that is implicit (for norms and norm instances at the same level) is not exhaustive nor flexible enough. Also, the operationalisation is descriptive rather than transition based, and this is a problem for the translation into an algorithmical solution. We will address these issues in the formal semantics we will present in Chapter 6.

5.1.5 Contributions and limitations of our language for contracts

In the previous subsections we have described a language for contracts, its layers and its operational semantics. This language had a grounding in XML and was used successfully in several use cases, targetting real-world applications based on web-services: car insurance, security certification and aerospace aftermarket[Jakob et al. 2008].

The language was expressive enough to describe, with a high level of detail, contracts based on human interactions, thanks to the capability of defining clauses with deontic expressions. Also, its formal grounding allowed to build a monitor based on augmented transition networks[Faci et al. 2008], as well as a model checker to infer properties such as safety and liveness of specific contracts[Lomuscio, Qu, and Raimondi 2009].

We provided relevant contribution with respect to the behavioural aspects of agents under agreements (performative extensions, interaction protocols). Such elements were implemented as agents enacting roles such as *Trusted Observer* or *Administrative Parties*.

However, there are still a few elements that we need in order to achieve proper governance in SOA from a monitoring perspective. First of all, the language should be generalised in order to isolate the normative layer (that is, the deontic part) to achieve a more lightweight language and more compatible with different domains. In fact, separating a language in as many layers as described is ideal in completely closed systems such as the ones demonstrated in the lifetime of the IST-CONTRACT project, but this is not the usual scenario in SOA.

Furthermore, our contract language is still lacking the capability of describing constitutive rules. We will see in the next section how we advanced in order to tackle these shortcomings by achieving a more compact language with added expressitivity features.

5.2 The ALIVE framework

The goal of the ALIVE Project[9] was to merge the state-of-the-art in organisational and normative modelling with the latest developments in service-oriented computing, using model-driver software engineering as the main tool[Alvarez-Napagao et al. 2009]. Meta-models for several levels (Service, Coordination and Organisation, see Figure 5.14), were built with a special focus on formal grounding and automatically transformed into a set of tools aimed at developers and stakeholders by means of meta-model transformations.

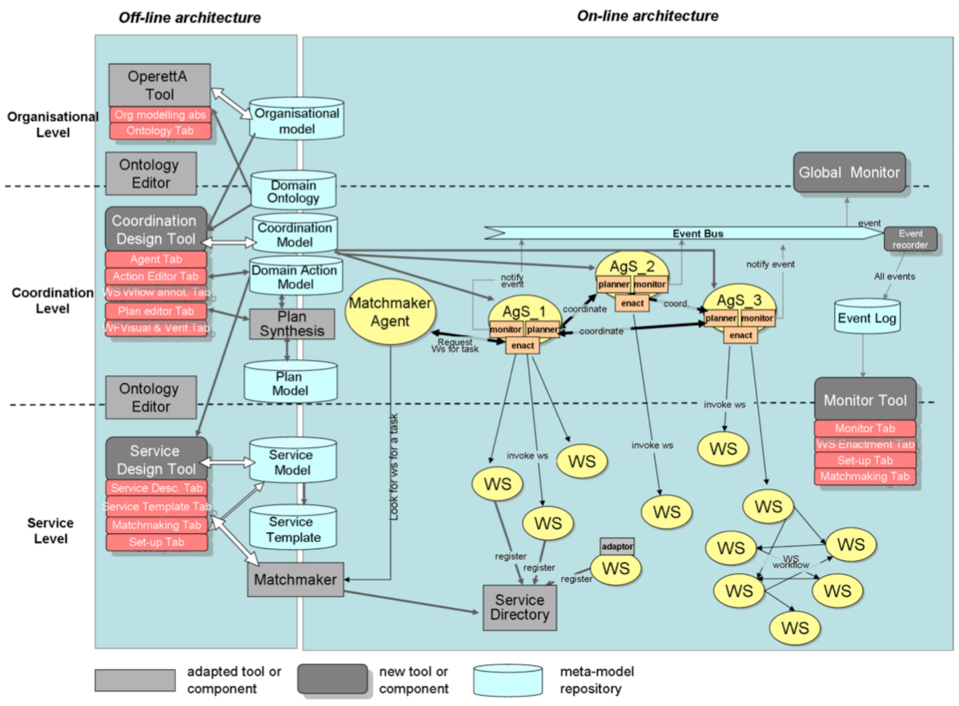


Figure 5.14: *ALIVE Multi-Level Architecture*

❧

This separation of abstraction layers allows the developer to specify, in a separate way and for a set of particular processes, *how* can things be done at the lower level of

[9]The ALIVE Project was a project (FP7-215890) funded by the European Commission within its 7th Framework Programme for RTD. The work presented in Section 5.2 was developed during the lifetime of the project.

services (Service Level), *what* high level goals can be achieved by a rational agent or a group of agents through joint actions (Coordination Level), and *why* should certain goals and norms be followed or taken into consideration (Organisational Level). The ALIVE Framework was designed to allow designing an organisation by mapping levels into existing software components (web-services or agents) in a bottom-up fashion, or by automatically deploying web-services or agents according to a fresh organisational specification, a top-down approach.

5.2.1 The ALIVE meta-model for norms

Due to the requirements of the project, we had to take into account multiple layers of abstraction in order to build the foundations of the conceptual framework of ALIVE[10]. For a deep understanding of the internals of each layer, please refer to [Aldewereld and Dignum 2010b; Aldewereld et al. 2010a; Alvarez-Napagao et al. 2009; Lam et al. 2009]. The meta-modelling approach, in which there are no true root elements such as e.g. in XML, allows for a effective decoupling of levels and even among layers of the same level. Therefore, we can isolate the relevant part for the purpose of this chapter: the Normative Structure, which is one of the parts of the Organisation Meta-model (see Figure 5.15)[11].

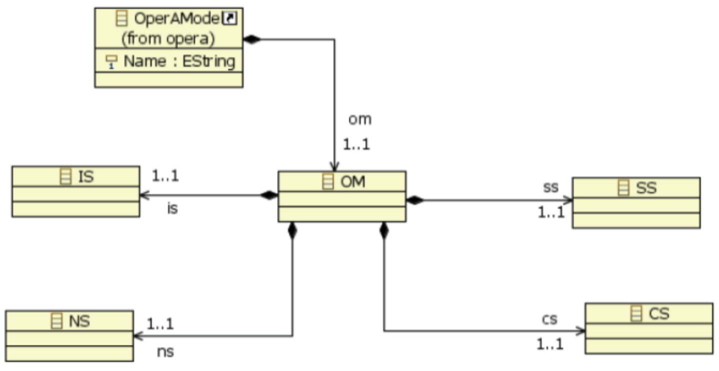


Figure 5.15: *Top-level elements of the organisation meta-model in ALIVE*

❧

[10]My contribution to the work presented in this section was twofold: a continuous involvement in the design of the Organisational Structure in general terms, esp. in the design of the Normative Structure by merging concepts from the IST-CONTRACT language with other approaches, and the design and the implementation of the operationalisation of several core concepts. These contributions were made along with Huib Aldewereld, Frank Dignum, Virginia Dignum and Javier Vázquez-Salceda.

[11]The other parts are: 1) the Social Structure, which contains roles and relationships between roles, 2) the Interaction Structure, which specifies scenes and transitions between scenes, and 3) the Communicative Structure, which contains domain ontologies used accross the Organisation Meta-model and the rest of layers.

The meta-model for the Normative Structure (NS), therefore, specifies the norms applied within the rest of the ALIVE Framework, including live components such as coordination-level agents or service-level services. The components of the NS can be seen at Figure 5.16.

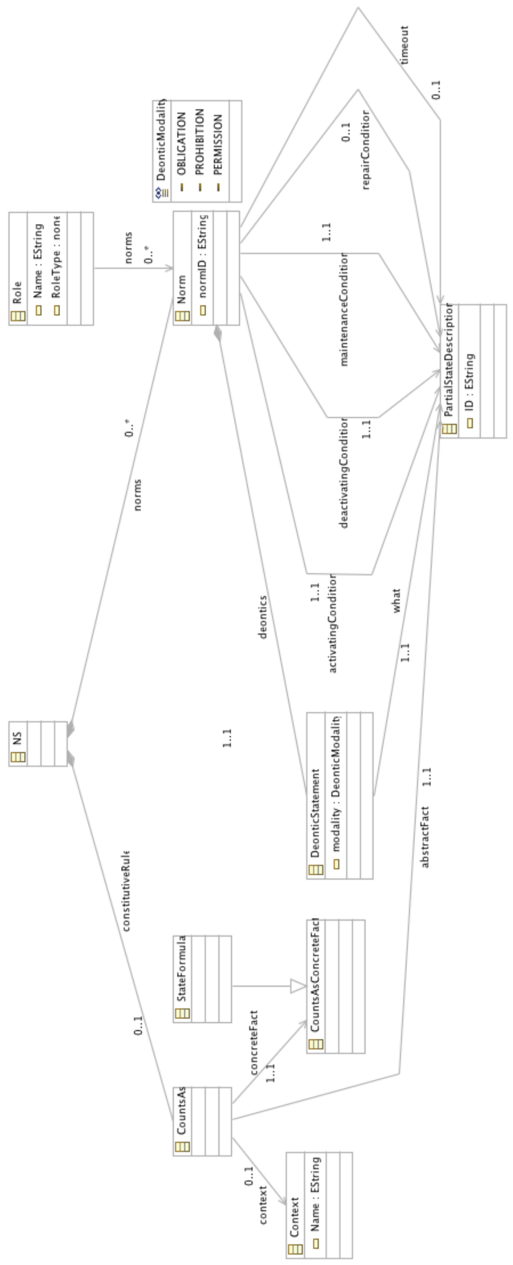


Figure 5.16: *Main components of the Normative Structure*

❧

IST-CONTRACT element	ALIVE element
Clause	**Norm**
ClauseID	**normID**
EndCondition	**deactivationCondition**
ExplorationCondition	**maintenanceCondition**

Table 5.4: *Mapping of concepts: IST-CONTRACT vs ALIVE Framework*

❧

The first component that stands out with respect to the IST-CONTRACT language is the *CountsAs* element, aggregated by NS as *constitutiveRules*. This is an addition over previous proposals and contains the constitutive definitions of the normative context. Each counts-as rule contains an abstract fact and a context that trigger the counts-as rule, and the definition of a concrete fact that will hold if and only if the rule is triggered.

The second element that can be aggregated under the label *norms* by the NS is *Norm*: a mapping of the *Clause* model presented in Section 5.1 with some variations. The most relevant of these variations is the disappearance of an explicit concept representing the deontic modality of the norm. This change was motivated by the fact that our language, thanks to the different (activating, maintenance, expiration) conditions, already allowed us to model obligations, prohibitions and permissions implicitly. This is elaborated further in Chapter 6 and Appendix A.

The correspondence between IST-CONTRACT clauses and ALIVE norms follow the pattern in Table 5.4, so the abstract norm tuple of Section 5.1.4 would look like:

$$\langle Modality, ActivationCondition, MaintenanceCondition, \\ DeactivationCondition, Who\rangle$$

Additionally, there are two relevant new elements: a *repairCondition*, used to specify repairing actions or sanctions that are supposed to bring a norm instance from a violated back to a “normal” state; and a *timeout*, that specifies an amount of time t such that if a norm instance has been violated for a period of time t, the instance is considered *failed*. The motivation for these two additions will be analysed extensively in Section 6.4, but as a summary we can say at this point that they allow a considerable improvement on the life-cycle of the IST-CONTRACT operational semantics.

From a structural point of view, an element of the model that is of great importance is *PartialStateDescription* (see Figure 5.17). This allows implementing count-as rule conditions, as well as the conditions that form part of each norm, in either propositional, first-order, or computational tree logic. A special kind of partial state description is the *landmark*, which indicate ideal intermediate states that can be used as a guidance in order to fulfill organisational goals. In ALIVE, landmarks may be used to model relevant/critical states of the system interaction, and can be mapped to norms governing such interactions.

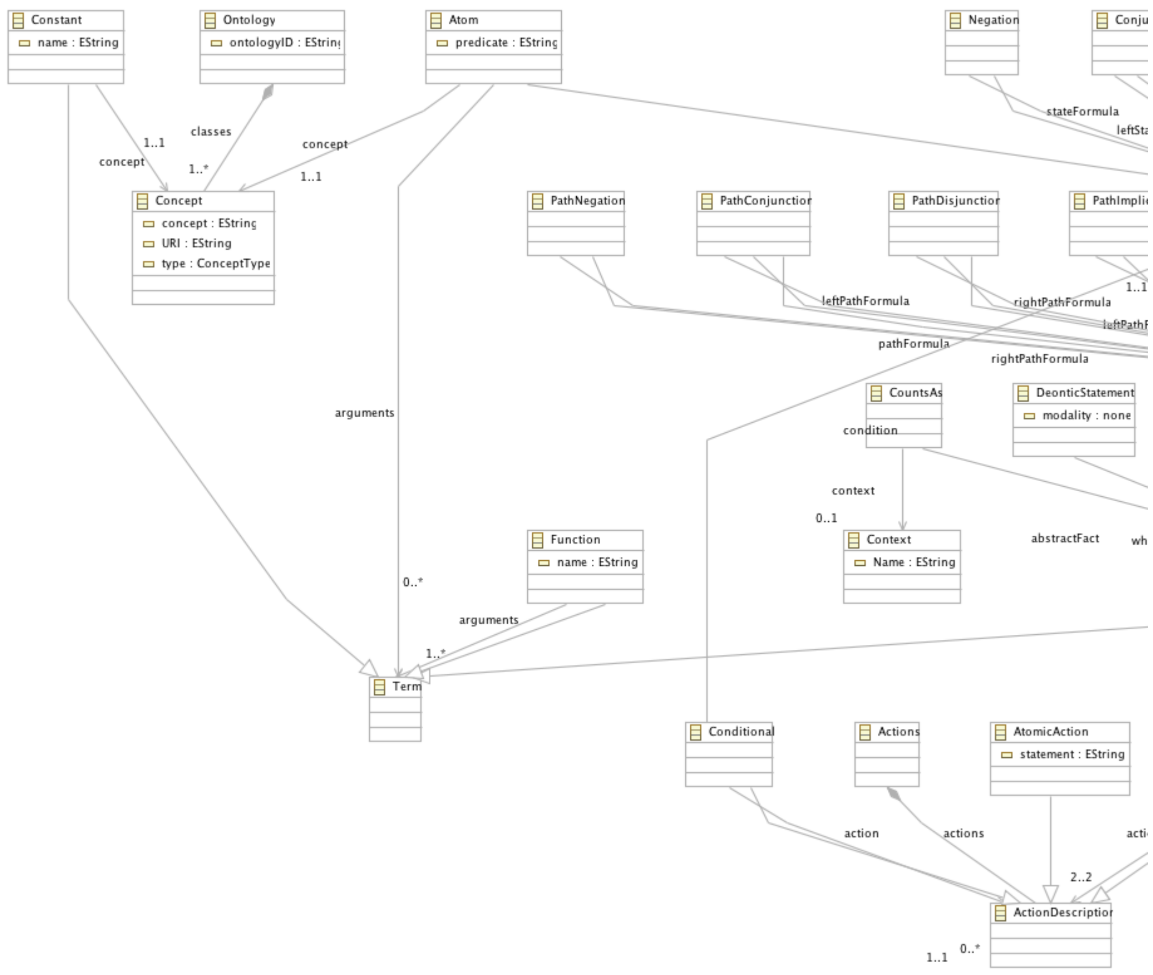
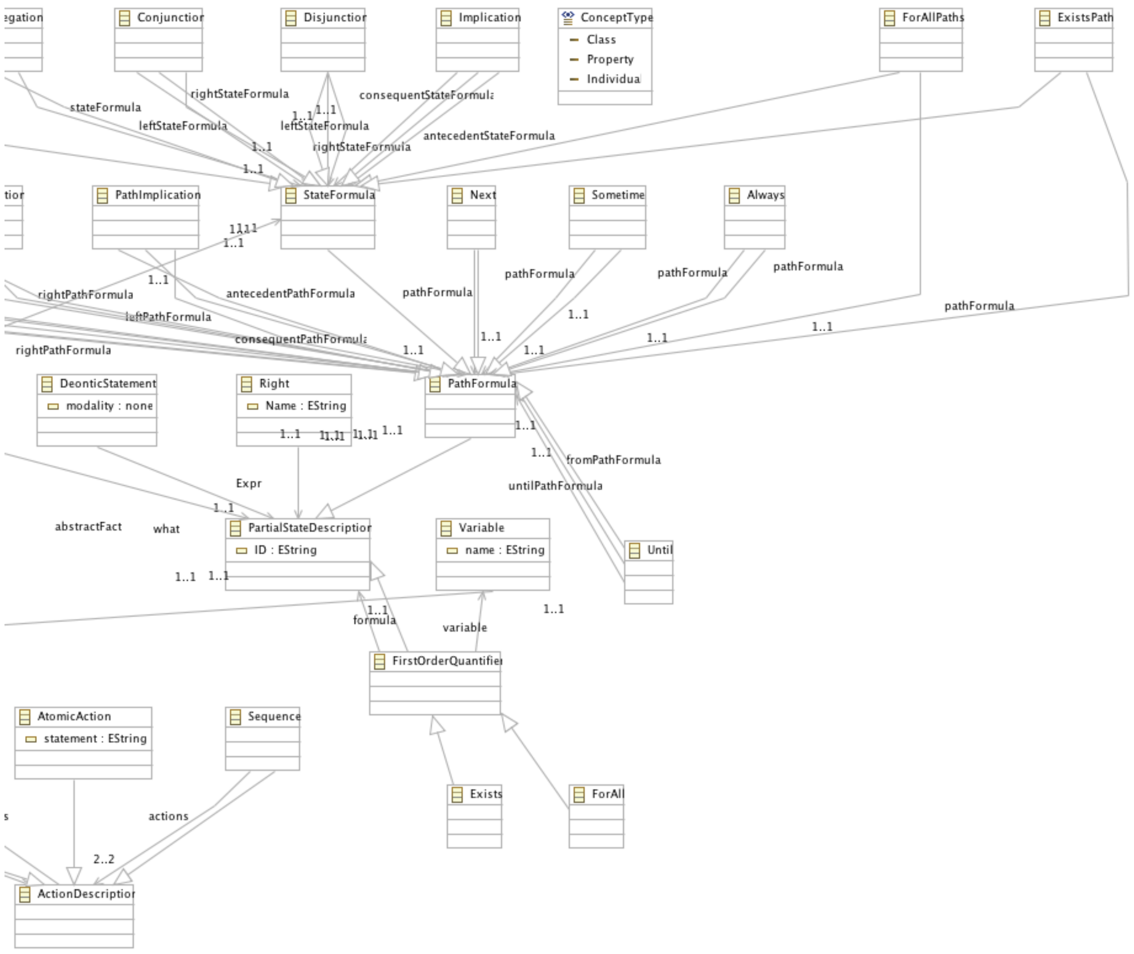


Figure 5.17: *Components of state descriptions in the ALIVE Framework*

❧



Figures 5.18 and 5.19 contain examples of a constitutive rule and a regulative norm that be modelled by using the proposed norm language in the ALIVE meta-model. For further and more elaborated examples of norms and other institutional abstractions modelled using ALIVE, see Section 9.1.3 and Section 9.2.3.1.

5.2.2 From events to monitoring

From the global architecture diagram (see Figure 5.14), there were several components and subcomponents directly involved in the monitoring process: the Global Monitor (organisational perspective) and the Local Monitor (individual agent perspective), both receiving *brute facts* from the Event Bus that acted as a broker for all the events generated both externally and internally[12].

[12]My contribution to the work presented in this section was the design and implementation of the event meta-model and the monitoring subsystems, being the coordinator of these tasks in the context of the ALIVE project. These contributions were made along with Huib Aldewereld and Javier Vázquez-Salceda.

Counts-as rule C: In a particular context where there is a certain power to marry people, two people are considered as married if they were declared (speech act) as such by a person who had that power.

Context C	$hasPower(A, declared(A, married(x, y))$
Antecedent C	$declared(A, married(B, C))$
Consequent C	$married(B, C)$

Figure 5.18: *Formal model for an example constitutive norm*

❧

Norm N: The repair company is obliged to notify the insurance company before April 10 if the report of the repair is ready. Otherwise, the insurance company has to report this violation to the repair company before one month after April 10.

Activation Condition N	$exists(RepairReport(car))$
Expiration Condition N	$sendRepairCompleted(car, DamageSecure, RepairCompany)$
Maintenance Condition N	$actualTime(t) \wedge t < April_1 0$
Repair Condition N	$reportViolation(RepairCompany, DamageSecure)$
Timeout N	$actualTime(t) \wedge t > May_1 0$

Figure 5.19: *Formal model for an example regulative norm*

❧

The key element upon which all these components were built was the Event part of the ALIVE meta-model. Our definition of event (there seems to be a lack of agreement on a common definition) was any *happening of relevance* that occurs within the *observable boundaries* of the system, where:

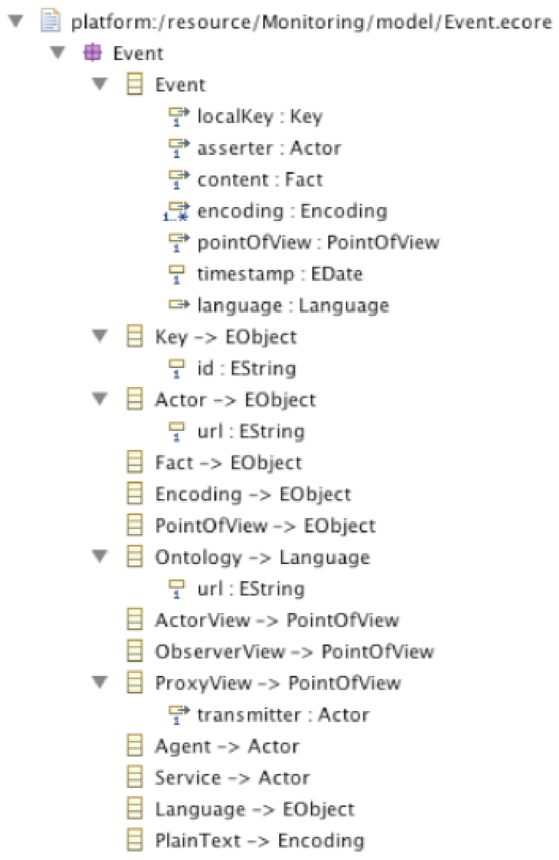
- A *happening* is a computable item: a change of state, or a fact related to a resource or an action;
- *Relevance* means that a specific item should be taken into consideration for a proper fulfilment of goals, objectives, or normative compliance; and
- The *Observable boundaries* of a system is the total set of items that can be "seen", limited by the technical and/or physical capabilities of the actors of the system.

Events should ideally be described by means of an ontology, in order to be understandable by the actors processing them. In ALIVE, being a layered architecture, events represented facts at different levels of abstraction (organisation, agent, service, external).

However, it was our objective to design the monitoring architecture in such a way that events were seen as abstract objects. For this reason, events were considered as abstract objects, independent of the information they carry. It was the responsibility of the actor processing an event to interpret the contents and the context of those contents.

5.2.2.1 *Event meta-model*

The ALIVE Event meta-model is shown in Figure 5.20. This meta-model contains all the elements necessary to model an event in the most possible general meaning of the concept, while representing additional attributes that may help not only understand the fact, but also unequivocally identify it and place the event in both time and location.



```
▼ platform:/resource/Monitoring/model/Event.ecore
  ▼ Event
    ▼ Event
          localKey : Key
          asserter : Actor
          content : Fact
          encoding : Encoding
          pointOfView : PointOfView
          timestamp : EDate
          language : Language
    ▼ Key -> EObject
          id : EString
    ▼ Actor -> EObject
          url : EString
      Fact -> EObject
      Encoding -> EObject
      PointOfView -> EObject
    ▼ Ontology -> Language
          url : EString
      ActorView -> PointOfView
      ObserverView -> PointOfView
    ▼ ProxyView -> PointOfView
          transmitter : Actor
      Agent -> Actor
      Service -> Actor
      Language -> EObject
      PlainText -> Encoding
```

Figure 5.20: *ALIVE Event meta-model*

❧

The main element, *Event*, represents the concept of event itself. It includes the following attributes:

- The *content*, which is the central attribute, and can be any kind of object.
- The *encoding*, which defines how the content is represented, e.g. as a plain text string, as an encrypted string (and thus defining as well how to decrypt it), or as

an XML object (and thus defining which is the XSD it is defined upon).

- The *language*, defining how to interpret the encoded content. In most cases the language is an ontology.
- The *asserter*, that is, the actor that stated the content of the event, e.g. an agent, a service, or an architectural component.
- The *local key*, which is a unique key created by the asserter. The combination of the asserter plus the local key has to be unique in the system.
- The *point of view* of the asserter with respect to the asserted fact. This attribute should describe how the asserter got conscience of the fact. In most cases, this point of view is one of the following: the asserter was the direct responsible of the happening, the asserter observed the happening, or the asserter acts as a proxy or re-transmitter of a third party responsible for the fact.
- The *timestamp* identifying the exact point in time in which the asserter became aware of the fact.

5.2.2.2 *Monitoring*

Regardless of whether a monitor is a local monitor or a global monitor, there are common properties that it has to implement.

First of all, a monitor may execute a set of rules to trigger events, e.g. send a notification of the reception of a certain event. These rules define the behaviour of a specific instance of a monitor. Our implementation was based on a rule engine – *Drools* – to manage and execute these rules.

The definition of these rules was based on facts to be matched, and thus a set of rules implicitly defined a set of facts to be kept under surveillance. Each monitor was then responsible for extracting this set of relevant facts and for subscribing them to the Event Bus.

Although a monitor may be configured to communicate directly with actors or components, monitors are also possible providers of events. Therefore, every monitor in the system was connected to the Event Bus.

Finally, every monitor uses one or more interface to communicate with actors and with external or internal components. Considering that a monitor is not only a receiver but also a sender of events, these interfaces have to work in both ways. We solved this by defining a subscribe mechanism for each interface – *push* rather than *pull*.

Figure 5.21 presents the generic monitor architecture, taking into account the constraint specified in the previous paragraphs. Regardless of the specific implementation details for each type of interface, we defined a generic API to be exposed for every actor or component that interacted with a monitor (see Figure 5.22).

- *initialize* and *updateRules* are methods to be called by the component responsible for creating and configuring the monitor instance. These two methods will define the active set of rules and thus determining the behaviour of the monitor.

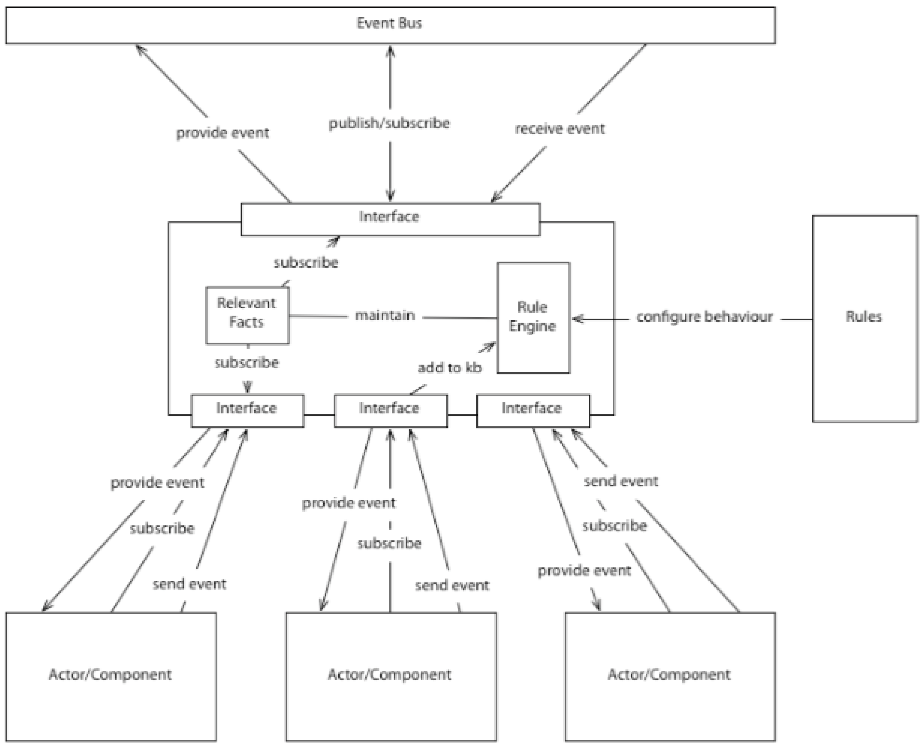


Figure 5.21: *ALIVE Monitoring Architecture*

❧

- *subscribe* is a method public to any component willing to receive certain notifications from the monitor, by specifying a list of facts and an endpoint. This endpoint will be used by the monitor to asynchronously send the subscribed events back. The specific way of handling these notifications will depend on the implementation of the interface. This method returns a session representing the subscription.
- *sendEvent* can be used by a component to notify an event to the monitor.
- *cancelSubscription* can be used by a component to remove the correspondent subscription and therefore indicating the wish to stop receiving notifications.

5.3 Conclusions

In this chapter we have presented a language for declaring norms, focusing on its adequacy for enabling governance capabilities in distributed systems, namely both SOA and multi-agent systems. We have done it incrementally, showing in the first place how we designed a language specifically tailored for compositions of web-services that are capable of communicating in the same domain and using the same behaviour protocols. This first version of the language allows a high level of expressivity with respect

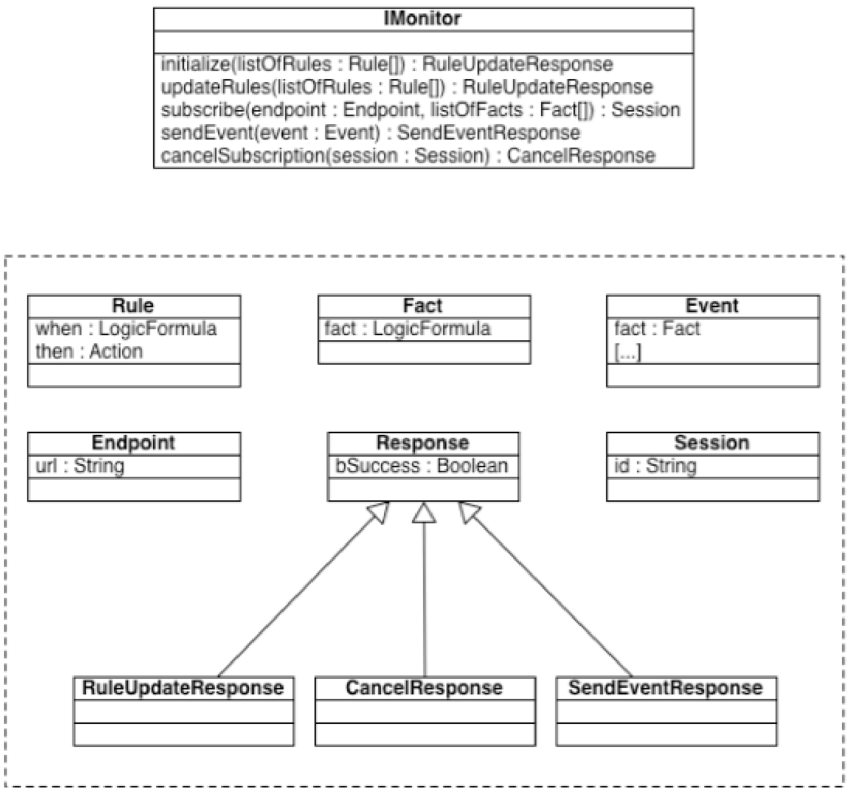

IMonitor
initialize(listOfRules : Rule[]) : RuleUpdateResponse
updateRules(listOfRules : Rule[]) : RuleUpdateResponse
subscribe(endpoint : Endpoint, listOfFacts : Fact[]) : Session
sendEvent(event : Event) : SendEventResponse
cancelSubscription(session : Session) : CancelResponse
Rule
when : LogicFormula
then : Action
Fact
fact : LogicFormula
Event
fact : Fact
[...]
Endpoint
url : String
Response
bSuccess : Boolean
Session
id : String
RuleUpdateResponse
CancelResponse
SendEventResponse

Figure 5.22: *Monitor generic interface*

❧

to regulative norms, by the use of deontic expressions. We have shown how such a language can be formally grounded.

Later in the chapter, we have presented an evolution of the pure normative part of the language in the form of a meta-model, adding constitutive rules to the language and abstracting the language from specific domains and agent behaviours.

The language in the latter form can be considered as lightweight enough for its use in different kinds of distributed systems, and therefore it is already suitable for describing norms. However, we still need to define a operationalisation more fitting for practical reasoning in real-time, and such operationalisation should include a formal grounding for the constitutive part. We have presented the monitoring architecture from an infrastructural perspective, but we have saved the details on the formalisation of the operationalisation for the following chapters: in Section 6.3 we will deal with the latter issue before concentrating on the full operational semantics in Section 6.4 and their implementation in Chapter 7.

CHAPTER 6

Formalising regulative and constitutive norms

'Take nothing on its looks; take everything on evidence. There's no better rule.'

Great Expectations
CHARLES DICKENS

Until recently, most of the work on normative environments works with norm specifications that are static and stable, and which will not change over time. Although this may be good enough from the social (institutional) perspective, it is not appropriate from the agent perspective. During their lifetime, agents may enter and leave several interaction contexts, each with its own normative framework. Furthermore they may be operating in contexts where more than one normative specification applies. So we need mechanisms where normative specifications can be added to the agents' knowledge base at run-time and be practically used in their reasoning, both to be able to interpret institutional facts from brute ones (by using constitutive norms to, e.g. decide if killing a person counts as *murder* in the current context) and to decide what ought to be done (by using regulative norms to, e.g. prosecute the murderer). Our proposal is to use production systems to build a norm monitoring mechanism that can be used both by agents to perceive the current normative state of their environment, and for these environments to detect norm violations and enforce sanctions. Our basic idea is that an agent can configure, at a practical level, the production system at run-time by adding abstract organisational specifications and sets of counts-as rules.

In our approach, the detection of normative states is a passive procedure consisting in monitoring past events and checking them against a set of active norms. This type of reasoning is already covered by the declarative aspect of production systems, so no additional implementation in an imperative language is needed. Using a forward-chaining rule engine, events will automatically trigger the normative state - based on the operational semantics - without requiring a design on *how* to do it.

Having 1) a direct syntactic translation from norms to rules and 2) a logic implemented in an engine consistent with the process we want to accomplish, allows us to decouple normative state monitoring from the rest of the agent reasoning, and thanks to this decoupling we can develop a norm monitor that can be used both by agents in the system to keep track of their normative situation, and by the institution to enforce norms. The initial set of rules we have defined is the same for each type of agent and each type of organisation, and the agent will be able to transparently query the current normative state at any moment and reason upon it. Also this decoupling helps building third party/facilitator agents capable of observing, monitoring and reporting normative state change or even enforcing behaviour in the organisation.

In this chapter we present a formalism for the monitoring of both regulative (deontic) and substantive (constitutive) norms based on a reduction from deontic logics to Linear Temporal Logic. More concretely, in Section 6.1 we propose an operationalisation of constitutive rules, Section 6.3 introduce a solution for dealing with norm instances, and Section 6.4 shows our operational semantics for regulative rules.

6.1 Constructing social reality

As already discussed in Section 1.3, there is a trend in service-oriented architectures to use coordination mechanisms in a purely hardcoded way. On the other hand, in many attempts of properly implementing regulated complex systems, organisational specifications are generally too abstracted from actual practice [Aldewereld et al. 2010a; Dignum and Vázquez-Salceda 2005; Dignum 2004]. This creates a distinct gap between the ontology of the organisation (containing abstract concepts such as "means of transport") and the ontology of the implementation (containing concrete concepts such as "trucks"; often domain and/or implementation dependent).

Therefore, there is a conflict in the way regulation is handled by the different perspectives. Normative abstractions bring increased stability over time and, thanks to effective separation of concerns, higher flexibility. However, because norms are usually seen as too abstract, it is difficult to relate abstract concepts of the design level to concrete actions and events of the implementation level. In this section we will show how to use constitutive (*counts-as*) rules, as defined in Section 2.1, to help bring closer both levels of abstraction.

In this section, we will use an example from one of the ALIVE Project use cases. In Netherlands, there is a nation-wide set of procedures for crisis management exists called GRIP (*Gecoördineerde Regionale Incidentbestrijdings Procedure*, Coordinated Regional Incident-Management Procedure). These procedures define the appropriate response to a crisis in terms of the required coordination, control, and flow of information based on the severity of the disaster. There are 4 levels, where GRIP-1 is the lowest, e.g., a large car accident or a small local fire, and GRIP-4 is the highest, e.g., a terrorist attack or large-scale flooding. These kinds of contexts require the ability to reason about different links between the abstract organisational concepts and concrete concepts used in practice (e.g., would scaling from one GRIP level to another provide additional means to solve the crisis).

6.2 The different meanings of constitutive rules

In Section 2.1 and Section 3.1 we already discussed that in [Searle 2009] counts-as rules are the foundation of institutions, and thus can effectively *constitute* social reality. Counts-as rules add institutional meanings to brute (ontologically objective) facts.The use of counts-as rules as a means to link abstract with concrete concepts in MAS has already been explored [Aldewereld 2007; Grossi 2007; Grossi et al. 2006]. An special case of proposal is [Jones and Sergot 1996], in which counts-as rules can be seen as subsumption relations:

> *"There are usually constraints within any institution according to which certain states of affairs of a given type count as, or* are to be classified as*, states of affairs of another type." [Jones and Sergot 1996]*

This is one of several meanings associated to counts-as rules: the *classificatory* view [Grossi, Meyer, and Dignum 2008].

One problem of counts-as rules seen as classificatory rules applied to distributed systems is that counts-as rules allow agents to reason about the links between abstract and concrete concepts (the γ_1 and γ_2 parts of the rule, respectively) based in a certain context (the C part of the rule), but if the meaning of such concepts or the contexts themselves change over time there are scenarios where problems may arise, such as our crisis management use case.

For example, the fact that an *army truck* counts-as a *means of public transport* is not always true, but only in the context of a large scale evacuation being carried out. In fact, it can be one of the counts-as rules that defines what a *large scale evacuation* is, e.g. the fact that army trucks are being used means that the evacuation must be a large-scale evacuation.

In this example we can see the other faces of counts-as rules: the face of a counts-as rule declaring something being constituted, on one hand, and the face of *some event constituting a context* [Grossi, Meyer, and Dignum 2008].

Scenarios such as crisis management, in which roles play an important part, counts-as rules are very important. They allow to effectively separate the high-level specification, e.g. organisational structure and regulative rules, while allowing particular low-level details to be constantly changing. For example, agents should know that army trucks are not available as means of transport when a burst water pipe floods a street, but are available when the context becomes that of a large scale evacuation after a dam has been breached.

We argue then that a system using counts-as rules should take into account not only the classificatory interpretation, but all of them. In order to do that we will treat constitutive contexts as first-class objects of our design, and use set relations to implement the dynamics of the counts-as various aspects.

6.2.1 Representing counts-as rules

As mentioned in Chapter 5 there is a need to explicitly represent the relations between the abstract and concrete concepts which can be used by agents in their reasoning. Moreover, since these relations between concepts are dependent on the context in which that relation is evaluated, the definition of the context of those relations needs to be explicit as well. This can be done with counts-as statements, which have three different readings, summarised in Table 6.1 [Grossi, Meyer, and Dignum 2006].

“It is always the case that large scale fires *count as* happenings with severe consequences to the general safety"	**Classificatory**
“In normative system Γ large scale fires *count as* disasters"	**Proper Classificatory**
“In normative system Γ, happenings with severe consequences to the general safety *count as* disasters"	**Constitutive**

Table 6.1: *Three notions of counts-as.*

❧

In this table, based on our use case, we can find three counts-as statements which provide different semantics. The first statement is a universal classification, whereas the second statement defines a contextual classification. The third statement provides a meaning that is not found in the other two: there is social reality being constituted, because the rule is defining a context in which this counts-as rule will hold.

Counts-as has the ability to change the world, not in the sense that it affects ontological objectivity – physical reality; it makes no sense to express that “children of the age of 2 *counts-as* writers”, since 2-year old children are physically unable to write. Stating it does not make them able to. Instead, counts-as adds institutional semantics to ontologically objective facts. Counts-as rules does not change what people can or cannot do physically, but it does change what people are allowed or entitled to do institutionally.

6.2.2 Agent reasoning with Counts-as rules

In complex scenarios such as in crisis management, the meaning of the concrete events may vary from context to context. In the GRIP-1 level – e.g. a car crash – the meaning of the institutional fact *sufficient coordination* means something different than in a GRIP-3 level situations – e.g. a flooding. While in the GRIP-1 level people can verbally communicate to coordinate the efforts of the different parties involved – police, medics, firemen – counts as *sufficient coordination*, in more pressing situations a larger coordination structure is required. The manner in which the coordination is managed differs between these scenarios: whereas in the car crash the orders can be done verbally between parties, coordinating in a large scale flooding requires some form of recordable communication channel.

Therefore, it is clear that the existence of consitutive rules affect the way agents behave. Consider, for example, the following norm: *Crisis handlers need to inform their superiors through adequate measures*. This norm is abstract in its description that crisis handlers – e.g., police officers or firefighters – need to use the appropriate channels to inform their supervisor. In a simple scenario, like the car accident mentioned earlier, a simple verbal *ok* to the coordinator on site will suffice. However, in more complex situations, it might be required to keep a log of all communication between the parties involved and a more complex action, e.g. acknowledging orders from your superior via a PDA might be required.

In the GRIP-1 level, agents can choose between informing their superiors via a verbal acknowledgement or using their PDA to acknowledge the orders given. Either of these events can be classified as doing an $inform$ action. Subsequently, it can be derived that doing an $inform$ action makes the agent fulfill the norm. In the GRIP-4 scenario, however, this is not the case. While a verbal *ok* to your superior still counts-as an $inform$, the norm specifies that all communication should be traceable, which the verbal inform is not. The normative counts-as relation between the actions and whether the norms are fulfilled has changed in this scenario, and the agents should adapt accordingly.

Note that the classifications on the lower level (between events and actions) can also dynamically change in a domain. While in our current example the verbal *ok* does not count as a traceable inform, there are contexts where that event would count as that kind of action, e.g., when the verbal communication is overheard by a trustable third-party.

In short, the links between the abstract and concrete parts of counts-as rules explicit allow the agents to reason about different contexts and changing contexts. By changing contexts, the normative (deontic) status of the agents changes. This enables expanding the reasoning of the agents by allowing them to find other means to reach their goals. Having an explicit representation of such links not only helps bringing together the levels of abstraction, but also give entity and meaning to the different context and allow reasoning about the dynamics of pertaining to certain contexts.

In our crisis management scenario, an agent might change the context from being a local incident to a regional disaster by starting to communicate through a central control unit. This choice of communication implicitly involves other parties in the disaster which might constitute different counts-as rules and possibly different norms.

6.2.3 Handling of dynamic contexts

We have seen in this chapter that the constitutive counts-as rules define the social context in which the counts-as holds. Generating a static representation triggering rules for the dynamics of contexts from constitutive specification, taking into account the domain of the variables being involved, could prove unbearable at a computational level. For example, in the case where an agent wants to decide whether a scale-up from GRIP-2 to GRIP-3 is required. To deal with this inefficiency at runtime, we consider contexts

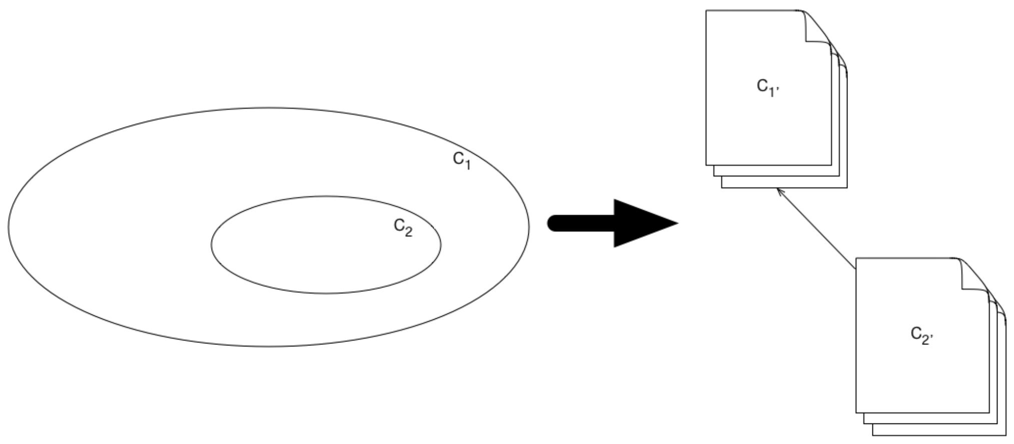


Figure 6.1: *Context subsumption.*

❧

to only have their unique counts-as rules (the rules that are not part of any other context). But this requires a proper handling of the occurrence of context subsumptions, i.e. context A being sub-context of context B; and context overlap, i.e. a non-empty intersection between the scopes of context A and B.

Any domain contains a number of social contexts defined by constitutive counts-as rules as mentioned above. These constitutive counts-as rules define the classifications that *only hold for that context*. Global classifications are considered to be part of the universal context, which subsumes each defined social context (i.e., all defined contexts are a *sub-context* of the universal context).

To deal with subsumed contexts and to allow for quick reasoning about what makes contexts unique, we limit the counts-as rules in a context to only those rules which are not contained in any of its *parent-contexts*. Then, by using context inheritance relations we specify that all the counts-as rules that hold in a context are those contained in its specification *and* any contained in the specification of its parents. Figure 6.1 shows the subsumption of context C_2 by context C_1; for instance, the social context of the GRIP procedures (C_2) being a sub-context of the social context of crisis management organisations (C_1). This basically means that the worlds in the context of GRIP are a 'refinement' of the worlds in the context of crisis management organisation; that is to say, these worlds adhere to both the classifications made by the parent context as well as to the classifications specified by the specific GRIP scenarios. Therefore, in a world in the social context of crisis management organisation, all counts-as rules of $C_{1'}$ apply, but in worlds in the social context of GRIP apply both the counts-as rules from $C_{1'}$ and $C_{2'}$. It is then easy to see that what makes the GRIP context different from the global context by looking at just the rules specified in C_2'.

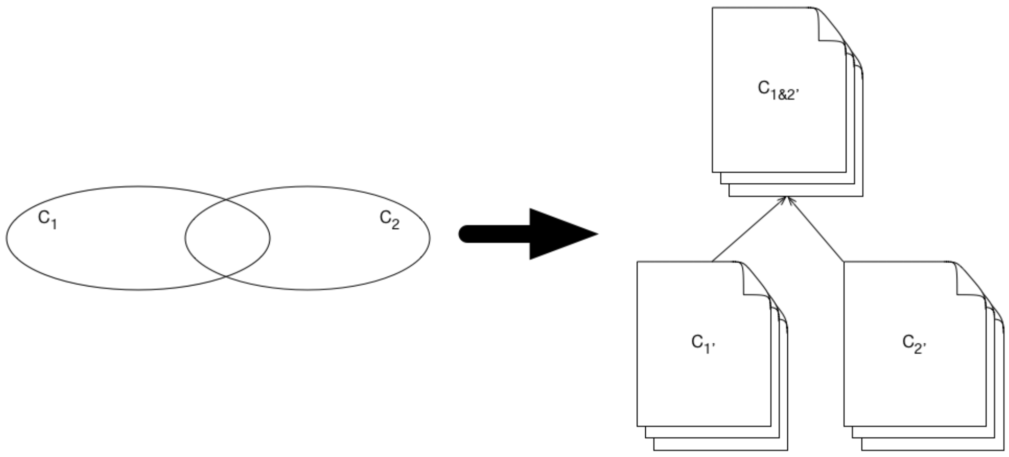


Figure 6.2: *Context overlap.*

❧

Similarly, we can deal with overlapping contexts. Take, for example, the different GRIP-levels; each are a specification of the crisis management situation at a different level of severity, but they all contain elements that remain the same between them; e.g., ambulances counts-as means of evacuation in both GRIP-2 and GRIP-3. There are, however, distinctions between the separate levels as well; e.g., army trucks count-as means of evacuation only in GRIP-3, not in GRIP-2. In Figure 6.2 it is visualised how contextual descriptions for C_1 and C_2 are split: a) a new shared, parent context (shown as $C_{1\&2}$ in the figure) that contains all counts-as rules that are shared between contexts C_1 and C_2, b) two distinct sub-contexts (shown as $C_{1'}$ and $C_{2'}$ in the figure) which contain the counts-as rules that make each original context distinct.

By this split it now becomes fairly easy to determine the differences between contexts C_1 and C_2: one looks at the specific rules for each in $C_{1'}$ and $C_{2'}$, respectively. Similarly, it is easy to determine the similarities between contexts C_1 and C_2 by looking at their shared parent-context $C_{1\&2}$.

Using this manner of reasoning with context overlap and context subsumption, we implemented the aspects of counts-as which are described in Section 6.2. However, we first look at how to implement norms with a production system in general in Section 7.3.1.

6.3 Dealing with norm instances

A relevant issue already discussed in Section 2.1 that is somehow missing in general in the literature is a clear separation between an abstract norm and a particular (contextual) instantiation of the norm. This problem was already discussed by Abrahams and Bacon in [Abrahams and Bacon 2002]: *"since propositions about norms are derived from the norms themselves, invalid or misleading inferences will result if we deal merely with the propositions* *rather than with the identified norms*[1] *that make those propositions true or false"*. This issue is not banal, as it has implications on the operational level: in order to properly check norm compliance, norm instantiations have to be tracked in an individual manner, case by case.

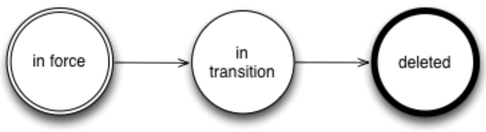


Figure 6.3: *Norm lifecycle*[2]

❧

We find useful, at this point, to stress the fact that the lifecycles of a norm, and of a norm instance, should be differentiated because they are different in essence. The lifecycle of a norm (see Figure 6.3) deals with its validity in the normative system: a norm is *in force* when it can be fully activated, monitored, and enforced; *in transition* when it is being removed and cannot be activated anymore, but the effects of past activations have to be tracked until their end; and *deleted* when the history of the norm is to be kept but it can have no further effect on the normative system. Therefore, such lifecycle is related to the concepts of promulgation, abrogation and derogation, out of the scope of this thesis[3]. On the other hand, the lifecycle of a norm instance deals with the fulfilment/violation of each particular instance.

The concept of norm instance lifecycle has been treated by different authors, e.g. [Abrahams and Bacon 2002; Cardoso and Oliveira 2010; Fornara and Colombetti 2009; Oren et al. 2009], but with no real consensus. Taking those interesting elements that would allow the management of norms with the concepts of activation, maintenance, fulfilment and reparation, a suitable norm lifecycle would be similar to the one based on the automaton depicted in Figure 6.4. A norm instance gets activated due to a certain activating condition and starts in an (A)ctive state, but if at some point a certain maintenance condition is not fulfilled, the norm instance gets into a (V)iolation state. If the norm instance is (A)ctive and a certain discharge condition is achieved, the norm gets (D)ischarged[4]. Usually reparations are not treated explicitly, but in our proposal we add the concept for completeness. If a norm instance is (V)iolated, fulfilling a reparation condition can bring it back to the (A)ctive state, but if the discharge condition occurs

[1]From now on, we will denote such identified norms as *norm instances*.

[2]This diagram, as well as ones that will appear later in the chapter, follows the standard notation of the state diagrams, where the arrow pointing from anywhere indicates an initial state and a double circle indicates a final state.

[3]Promulgation, abrogation and derogation of norms in a normative monitor are the main research issue that is tackled by Ignasi Gómez-Sebastià in his PhD thesis, which will be presented in January 2016. In the meantime the interested reader can check [Gómez-Sebastià and Alvarez-Napagao 2012].

[4]Note here that we assume the discharge condition to eventually happen.

while violated, only by fulfilling the same reparation condition (VD state) can the norm instance be (D)ischarged. It might be the case that a (V)iolated norm instance never gets repaired, so for safety we use a *timeout* condition[5] to make sure the norm instance is not alive forever and thus mark those permanent violations as (F)ailures.

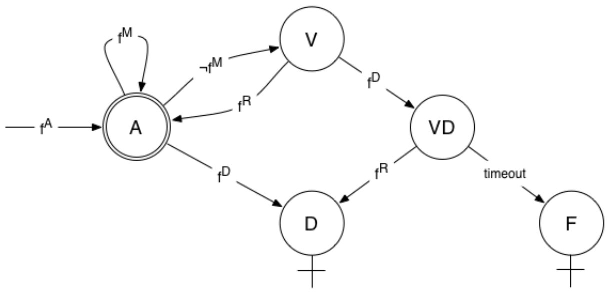


Figure 6.4: *Norm instance lifecycle with reparation and timeout handling*

❧

Once there is a norm life-cycle the question to answer is how to deal with it from an operational perspective. Abrahams and Bacon [Abrahams and Bacon 2002] solve this problem by means of *occurrences* of the predicates contained in the deontic operator, but there are cases in which this can be insufficient, e.g., when the obligation defines a deadline or its instantiation depends on contextual information. More recently, some works have been advancing in the direction of tackling this issue. For example, by treating instantiated deontic statements as first-class objects of a rule-based language [Cardoso and Oliveira 2009; Governatori 2005]. However, as these deontic statements are already implicitly identifying the norm instance, there is no explicit tracking of which elements of the domain are involved in fulfilling or violating. Other approaches declare the norm only at the abstract level and the tracking of the norm instance, and implicitly of the norm instance lifecycle, is purely done at the operational level [Alvarez-Napagao et al. 2011; Criado et al. 2010; Oren et al. 2009].

6.4 Formal Semantics

In this section we define the semantics of institutions as the environment specifying the regulative and constitutive norms and then discuss the formal semantics of our regulative framework. The results of this formalisation will then be used in Chapter 7 to describe the details of how an institution evolves over time based on incoming events, and how this impacts the monitoring process.

Through the rest of this chapter, we will use as an example the following simplified traffic scenario:

[5]The timeout condition is evaluated as starting at the point of time of violation.

1. A person driving on a street is not allowed to break a traffic convention.
2. In case (1) is violated, the driver must pay a fine.
3. In a city, to exceed 50kmh counts as breaking a traffic convention.

6.4.1 Preliminary definitions

In this section, we present a proposal for a deontic logic for support for norm instantiation via obligations parametrised by five states (conditions).

For the purpose of this formalization, we assume the use of a predicate based propositional logic language $\mathcal{L}_O$ with predicates and constants taken from an ontology O, and the logical connectives $\{\neg, \vee, \wedge\}$. The set of all possible well-formed formulas of $\mathcal{L}_O$ is denoted as $wff(\mathcal{L}_O)$ and we assume that each formula from $wff(\mathcal{L}_O)$ is normalised in Disjunctive Normal Form (*DNF*). Formulas in $wff(\mathcal{L}_O)$ can be partially grounded, if they use at least one free variable, or fully grounded if they use no free variables, i.e. only predicates and constants.

In this chapter we intensively use the concept of variable substitution. We define a substitution instance $\Theta = \{x_1 \leftarrow t_1, x_2 \leftarrow t_2, ..., x_i \leftarrow t_i\}$ as the substitution of the terms $t_1, t_2, ..., t_i$ for variables $x_1, x_2, ..., x_i$ in a formula $f \in wff(\mathcal{L}_O)$. Thus, $\Theta(f(x_1, x_2, ..., x_i)) \equiv f(t_1, t_2, ..., t_i)$. We will denote as $\vartheta_{(wff(\mathcal{L}_O),\mathcal{S})}$ the set of all possible substitution instances containing the variables in $wff(\mathcal{L}_O)$ and the terms in $\mathcal{S}$.

We denote the set of roles in a normative system as the set of constants R, where $R \subset O$, and the set of participants as P, where each participant enacts at least one role according to the ontology O.

As our aim is to build a normative monitoring mechanism that can work at real time, special care has been made to choose a norm language which, without loss of expresiveness, has operational semantics that can then be mapped into production systems. Based in our previous work and experience, our definition of *norm* in an extension of the *abstract norm* defined in the formalisation in Section 5.1.4 taking into account the ALIVE extensions of repair and timeout presented in Section 5.2.1:

Definition 9 (Norm) *We define a norm n as a tuple $n = \langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout \rangle$, where:*

- *α is the agent obliged to comply with the norm,*
- *f_n^A is the activating condition of the norm,*
- *f_n^M is the maintenance condition of the norm,*
- *f_n^D is the deactivation condition of the norm,*
- *f_n^R is the repair condition of the norm,*
- *$timeout$ is a fully-grounded formula that represents the upper-bound waiting condition for the reparation of a violation, taken into account of only after a violation and not before, and*
- *$f_n^A, f_n^M, f_n^D, f_n^R, timeout \in \mathcal{L}$.*

□

In order to create an optimal norm monitor it is important to know which norms are active at each point in time, as only those are the ones that have to be traced (inactive norms can be discarded from the monitoring process until they become active again). The *activation condition* f_n^A specifies when a norm becomes active. It is also the main element in the norm instantiation process: when the conditions in the activating condition hold, the variables are instantiated, creating a new norm instance[6]. The *deactivating condition* f_n^D defines when the norm becomes inactive. The *maintenance condition* f_n^M defines the conditions that, when no longer hold, lead to a violation of the norm. Finally, the *repair condition* f_n^R specifies when a norm instance stops being violated, as long as the *timeout* condition is not activated while the violation holds.

An example of a norm for the traffic scenario ("*A person driving on a street is not allowed to break a traffic convention*") would be formalised as follows:

$$\begin{aligned} n1 = \langle & Ag, \\ & \{enacts\_role(Ag, Driver) \wedge driving(Ag)\}, \\ & \{\neg crossed\text{-}red(Ag, L)\}, \\ & \{\neg driving(Ag)\}, \\ & \{fine\text{-}paid(100)\}, \\ & time(500)\rangle \end{aligned}$$

The activating condition states that each time an event appears where an individual enacting the $Driver$ role drives ($driving$), then a new instance of the norm becomes active; the maintenance condition states that the norm will not be violated while no traffic convention is violated; this norm has no deadline, it is to apply at all times an individual is driving; the norm instance deactivates when the individual stops driving[7]; the target of this norm is that we want drivers not breaking traffic conventions; finally the subject of the norm is someone enacting the $Driver$ role.

It is important to note here that, although our norm representation does not explicitly include deontic operators, the combination of the activation, deactivation and maintenance conditions is as expressive as conditional deontic statements with deadlines as the ones in [Dignum et al. 2004]. It is also able to express unconditional norms and maintenance obligations (i.e. the obligation to keep some conditions holding for a period of time). To show that our representation can be mapped to conditional deontic representations, let us express the semantics of the norm in definition in terms of conditional deontic statements.

Definition 10 (Deontic interpretation) *The deontic interpretation of a norm n, is:*

$$O_{f_n^R \leq timeout}([\alpha\ stit : f_n^M] \preceq f_n^D \mid f_n^A)$$

[6]One main differentiating aspect of our formalisation is that we include variables in the norm representation and we can handle multiple instantiations of the same norm and track them separately.

[7]Although the norm is to apply at all times an individual is driving, it is better to deactivate the norm each time the individual stops driving, instead to keep it active, to minimise the number of norm instances the monitor needs to keep track at all times.

□

The syntax of the operator proposed is similar to the obligation operator from other deontic logics, such as dyadic deontic logic and semantics of deadlines, but with important differences. While the $\leq$ used for $f_n^R \leq timeout$ corresponds to the deadline semantics found e.g. in [Dignum et al. 2004] or [Governatori et al. 2007] (if $timeout$ occurs, there is a permanent violation), the $\preceq$ used in $[\alpha\ stit : f_n^M] \preceq f_n^D$ should rather be read as "$[\alpha\ stit : f_n^M]$ should hold *at all times* at least until f_n^D". Also, the conditional notation $|$ used in dyadic deontic logic, which not always has clear semantics in terms of temporality, in the case of the operator proposed $O(A|B)$ should be read as "starting the moment B happens, A should happen" rather than simply "given B, A should happen"[8].

Therefore, the expression shown in Definition 10 is informally read as: *if at some point f_n^A holds, agent α is obliged to see to it that f_n^M is maintained until, at least, f_n^D holds; otherwise, α is obliged to see to it that f_n^R before $timeout$.* Note that in this informal reading we are not dealing with norm instances yet. How we address this issue, along with the semantics of this obligation operator, will be explained in Section 6.4.2. Following the example:

$$O_{fine\text{-}paid(100) \leq time(500)}([Ag\ stit : \neg crossed\text{-}red(Ag, L)] \preceq \neg driving(Ag) \mid driving(Ag))$$

informally read as: *if at some point Ag is driving, Ag is obliged to see to it that no red light is crossed until, at least, Ag is not driving anymore; otherwise, Ag has to pay a fine of 100 before the time is 500*. The semantics of this operator are presented in the rest of this section.

We define the *state of the world* s_t at a specific point of time t as the set of predicates holding at that specific moment, where $s_t \subseteq O$, and we will denote S as the set of all possible states of the world, where $S = \mathcal{P}(O)$. We will call *expansion* $F(s)$ of a state of the world s as the minimal subset of $wff(\mathcal{L}_O)$ that uses the predicates in s in combination of the logical connectives $\{\neg, \vee, \wedge\}$.

One common problem for the monitoring of normative states is the need for an interpretation of brute events as institutional facts, also called constitution of social reality[Grossi 2007]. The use of *counts-as rules* helps solving this problem. Counts-as rules are multi-modal statements of the form $[c](\gamma_1 \to \gamma_2)$, read as "in context c, γ_1 *counts-as* γ_2". In this thesis, we will consider a context as a set of predicates, that is, as a possible subset of a state of the world:

Definition 11 (Counts-as rule) *A counts-as rule is a tuple $c = \langle \gamma_1, \gamma_2, s \rangle$, where $\gamma_1, \gamma_2 \in wff(\mathcal{L}_O)$, and $s \subseteq O$.* □

A set of counts-as rules is denoted as C. Although the definition of counts-as in [Grossi 2007] assumes that both γ_1 and γ_2 can be any possible formula, in our framework we limit γ_2 to a conjunction of predicates for practical purposes.

[8] In some works in the literature, this is interpreted as "given B and as long as B happens, A should happen", while in other works it is interpreted in a closer way to our reading

Definition 12 (Institution) *Following the definitions above, we define an* institution *as a tuple of norms, roles, participants, counts-as rules, and an ontology:*
$I = \langle N, R, P, C, O \rangle$ □

An example of I for the traffic scenario would be formalised as follows:

$$\begin{aligned}
\mathbf{N} :=& \{\langle Ag, \{enacts\_role(Ag, Driver) \wedge driving(Ag)\}, \{\neg crossed\text{-}red(Ag, L)\},\\
& \{\neg driving(Ag)\}, \{fine\text{-}paid(100)\}, time(500)\rangle\}\\
\mathbf{R} :=& \{Driver\}, \mathbf{P} := \{Person_1\}\\
\mathbf{C} :=& \{\langle exceeds(D, 50), traffic\_violation(D), is\_in\_city(D)\rangle\}\\
\mathbf{O} :=& \{role, enacts\_role, driving, is\_in\_city,\\
& exceeds, traffic\_violation, paid\_fine,\\
& Person_1, role(Driver), enacts(Person_1, Driver)\}
\end{aligned}$$

6.4.2 Norm fulfillment and norm instance fulfillment

If we have to take into account the following issues[9]:

1. deontic statements do not express a norm, but rather the existence of a norm [Walter 1996]; and
2. in order to check the compliance of a norm, its particular instances must be tracked [Abrahams and Bacon 2002],

then we need to define the compliance of a norm based on the fulfillment of each of its instantiations. That is, a norm has been complied up to a certain time t if, and only if, each one of the instantiations triggered in times $t_i < t$ have not been violated, where violated means that there has been $\neg f_n^M$ before f_n^D ever happening.

In order to work with instances, we define a norm instantiation. A norm is defined in an abstract manner, affecting all possible participants enacting a given role. Whenever a norm is active, we will say that there is a *norm instance* n^θ for a particular norm n and a substitution instance θ.

Definition 13 (Norm instance) *Given a norm n and a substitution set θ, we define a norm instance n^θ as:*
$n^\theta = \langle \alpha, \theta(f_n^A), \theta(f_n^M), \theta(f_n^D), \theta(f_n^R), timeout\rangle$*, where:*

- *$\theta(f_n^A)$ is fully grounded, and*
- *$\theta(f_n^M)$, $\theta(f_n^D)$, $\theta(f_n^R)$ may be fully or partially grounded.*

□

The reason that $\theta(f_n^M)$, $\theta(f_n^D)$, $\theta(f_n^R)$ may be partially grounded is that the substitution instance that instantiates the norm – that is, θ such that $\theta(f_n^A)$ holds – is considered in our model to be the sufficient and necessary set of substitutions needed to fully ground f_n^A. It can be the case that the set of variables used in f_n^M, f_n^R and/or f_n^D is larger than the arity of θ. Let us suppose, for example, that the norm should be

[9] We have discussed these issues in more detail in Section 6.3.

instantiated at all times while it is *in force*, regardless of any contextual condition: in that case, $f_n^A = \top$. Therefore, we have to assume that a substitution instance θ' for f_n^M, f_n^R or f_n^D should fulfill: $\theta \subseteq \theta'$.

6.4.3 Semantics of LTL

In Chapter 4, we motivated the need for a formalisation allowing for grounding in both monitoring and planning contexts to allow both institutional and individual reasoning towards norms, in a way that the products of both kinds of reasoning are coherent and consistent between themselves at a formal (deontic) level.

While in Section 2.2 we saw that monitoring (verification of compliance) is usually formalised in different ways at distinct levels of complexity, the state of the art in planning is fairly more restricted. Popular high-level formalisms such as STRIPS, SOAR or HTN [Nau et al. 2003; Newell, Rosenbloom, and Laird 1987; Nilsson and Fikes 1970] are not expressive enough to deal with complex constraints mandatory for practical deontic reasoning such as temporal predicates. PDDL solves this issue [Fox and Long 2009; Gerevini and Long 2005] by explicitly adding the Linear Temporal Logic (LTL) operators.

In the field of research, PDDL is the most common language used in both formalisations and implementations, being the reference for benchmarks [Coles et al. 2012], and the capability of using temporal operators will prove decisive in order to build our formalisation. However, as we will see in Section 6.4.4, using LTL imposes some hard constraints. Unfortunately, there are no practical planners using temporal logics more complex than LTL such as CTL*, so these are constraints we will have to deal with. This is an approach also taken in related proposals in the context of socio-technical systems [Aldewereld, Dignum, and Meyer 2007; Riemsdijk et al. 2015].

In this section, we briefly present the main concepts of LTL needed to build our formalisation upon. LTL is built up from a finite set of propositional variables AP, the logical operators $\neg$ and $\vee$, and the temporal modal operators X and U. Formally, the set of LTL formulas over AP is inductively defined as follows:

- if p $\in$ AP then p is a LTL formula;
- if ψ and ϕ are LTL formulas then $\neg\psi, \phi \vee \psi, \mathbf{X}\psi$ *and* $\phi\mathbf{U}\psi$ are LTL formulas.

X is read as next and U is read as until. Sometimes, N is also used in place of X. Other than these fundamental operators, there are additional logical and temporal operators defined in terms of the fundamental operators to write LTL formulas succinctly. The additional logical operators, which can be derived by the primitive ones, are $\wedge, \rightarrow, \leftrightarrow$, true, and false. Following are the additional temporal operators.

- G for always (globally)
- F for eventually (in the future)
- R for release
- W for weakly until

An LTL model $M = (S, \Re, \pi)$ consists of a non empty set S of states, an accessibility relation $\Re$ and an interpretation function π for propositional atoms. A full path σ in M is a sequence $\sigma = s_0, s_1, s_2, \dots$ such that for every $i \geq 0$, s_i is an element of S and $s_i \Re s_{i+1}$, and if σ is finite with s_n its final state, then there is no state s_{n+1} in S such that $s_n \Re s_{n+1}$. We say that the full path σ starts at s if and only if $s_0 = s$. We denote the state s_i of a full path $\sigma = s_0, s_1, s_2, \dots$ in M by σ_i. Validity $M, s \models \phi$, of an LTL formula ϕ in a world state s of a model $M = (S, \Re, \pi)$ is defined as:

- $M, s \models p$ $\Leftrightarrow s \in \pi(p)$
- $M, s \models \neg\phi$ $\Leftrightarrow$ not $M, s \models \phi$
- $M, s \models \phi \vee \psi$ $\Leftrightarrow M, s \models \phi$ or $M, s \models \psi$
- $M, \sigma, s \models \mathbf{X}\phi$ $\Leftrightarrow M, \sigma_1 \models \phi$
- $M, \sigma, s \models \phi \mathbf{U} \psi$ $\Leftrightarrow \exists n > 0$ such that
  (1) $M, \sigma_n \models \psi$ and
  (2) $\forall\ i\ with\ 0\ \leq i < n\ it\ holds\ that\ M, \sigma_i \models \phi$

Validity on an LTL model M is defined as validity in all states of the model. If ϕ is valid on an LTL model M we say that M is a model for ϕ. General validity of a formula ϕ is defined as validity on all LDL models. The logic LTL is the set of all general validities of $\mathcal{L}_{LTL}$ over the class of LTL models.

The additional temporal operators R, F, and G are defined as follows:

- ϕ R $\psi = \neg(\neg\phi$ U $\neg\psi)$
- F ψ = true U ψ
- G ψ = false R $\psi = \neg$ F $\neg\psi$

6.4.4 Norm lifecycle

Although LTL as a formalism is suitable enough in terms of complexity for reductions to monitoring and planning scenarios, and therefore for practical reasoning from an institutional or individual perspective, there are intrinsic constraints that limit the expressiveness of the framework.

More concretely, the norm instance lifecycle proposed in Figure 6.4 cannot be expressed in LTL. As proved in [Tauriainen 2006], in order to reduce an automata to an LTL expression – and vice versa –, such automata has to be free of loops that involve more than one state, i.e. only cycles that start and finish in the same state and involve no second state are allowed.

This is an important constraint that prevents our model to have a loop between the (A)ctive and the (V)iolated states. In other words, if we want to use LTL, the lifecycle cannot have cycles that allow to go backwards. Therefore, for the purpose of our formalisation, we propose to adopt the more straightforward lifecycle shown in Figure 6.5.

The main difference with respect to the automata in Figure 6.4 is the handling of violations. As there is no way back to an (A)ctive state anymore, from a (V)iolation

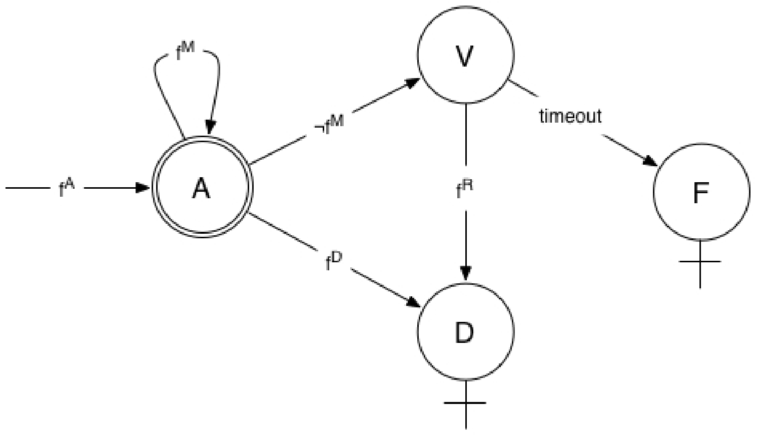


Figure 6.5: *Self-loop alternating automata-based norm instance lifecycle*

❧

state there are only two options: either to repair the norm instance and subsequently (D)eactivate it, or mark it as a (F)ailure if it has not been dealt with for a given amount of time. From an operational perspective, this issue can be worked around by allowing the norm-aware system to create more instances of the same norm if an instance is violated before a deactivation.

For an obligation to have a deontic effect, it is required that the activating condition actually happens at some future point. Additionally, either of the following three conditions should happen:

- The activating condition never occurs so the norm never gets activated.
- Always, between the activating and deactivation condition, the maintenance holds (reached “deactivated” state).
- Maintenance condition holds up to a point where it becomes false and then a violation is permanently raised. In addition, the repair condition occurs later (reached “deactivated” state) before timeout is reached.

In this way we approach most closely that the maintenance of $\theta(f_n^M)$ causes the $\neg viol(n^\theta)$. Thus, the deontic effect of an obligation can be described by the causal effect between the maintenance condition and a violation in Definition 14.

In order to give meaning to the fulfilment of a norm instance, we define a specific operator $\mathcal{O}$ with similar syntax to the abstract norm operator O. Let $M = \{S, \Re, \theta\}$ be an LTL model (using a predicate set $\mathcal{L}$ for the formation of LTL formulas and with θ as described in Section 6.4.1), $\pi =< s_0, s_1, s_2, \dots >$ a full path in M, and $viol(n^\theta)$ a predicate belonging to $\mathcal{L}$ representing the violation of a norm instance n^θ, we can establish the semantic relationship between the lifecycle of a norm instance and the fulfilment/violation of a norm as:

Definition 14 (Causal semantics for the operator $\mathcal{O}$)

$$\begin{aligned}
&M,\pi \models \mathcal{O}_{\theta(f_n^R)\leq timeout}([\alpha\ stit : \theta(f_n^M)] \preceq \theta(f_n^D) \mid \theta(f_n^A)) \\
&\qquad \equiv_{def} \\
&M,\pi \models \mathbf{G}(\neg\theta(f_n^A) \wedge \neg viol(n^\theta)) \vee \\
&\qquad \Big(\mathbf{F}\ \Big(\theta(f_n^A) \wedge [\forall\theta' : \theta'(\theta(f_n^M))\mathbf{U}\exists\theta'' : \theta''(\theta(f_n^D))]\Big) \wedge \mathbf{G}\neg viol(n^\theta)\Big) \vee \\
&\qquad \mathbf{F}\Big(\theta(f_n^A) \wedge [\neg viol(n^\theta)\mathbf{U}\exists\theta' : \neg\theta'(\theta(f_n^M))] \wedge \\
&\qquad\quad [\theta'(\theta(f_n^M))\mathbf{U}(\neg\theta'(\theta(f_n^M)) \wedge \mathbf{G}viol(n^\theta) \wedge (\neg timeout\mathbf{U}\exists\theta'' : \theta''(\theta(f_n^R))))]\Big)
\end{aligned}$$

□

The first line of the temporal formula says that the activating condition actually never happens and no violation is raised throughout the executional path. This case does not cause any change in the state of the system. The second line says that there exists some substitution for the activating condition in the future, and that always until a substitution raises an instance of the deactivation condition, the maintenance condition holds for all substitutions. No violation is raised throughout the executional path. This case terminates the norm in a state of *deactivation* (D). The rest of the lines in the formula imply that there exists some substitution for the activating condition in the future, and that at some later point a substitution makes the maintenance condition not hold, thus raising a violation (which remains thereafter). In addition, another substitution makes the repair condition happen at some future after the violation has occurred but before timeout occurs. The norm terminates in a state of *deactivation* (D).

The *failed* state (F), in which the timeout has occurred without the norm having realised the repair condition after a violation, is not described in the formula, since it is an "unwanted" state and should be avoided.

The lifecycle defined in Figure 6.5 can be seen as an transition automaton. Transition properties that define how the norm changes its status while events (world changes that modify the predicates' truthness) are occurring can be easily extracted. We are interested in directly representing these transitions as it is useful when dealing with monitoring of norms' status (see Chapter 7). The four states *active* (A), *viol* (V), *deactivated* (D), *failed* (F) are described in Definition 15:

Definition 15 (Norm lifecycle predicates)

$M,\pi \models \mathbf{X}active(n^\theta)$ *iff* $M,\pi \models (\mathbf{X}\theta(f_n^A) \vee active(n^\theta)) \wedge \mathbf{X}\ \nexists\theta' : \theta'(\theta(f_n^D))$

$M,\pi \models \mathbf{X}viol(n^\theta)$ *iff* $M,\pi \models active(n^\theta) \wedge \mathbf{X}\ \nexists\theta' : \theta'(\theta(f_n^M))$

$M,\pi \models \mathbf{X}deactivated(n^\theta)$ *iff*
$\quad (M,\pi \models active(n^\theta) \wedge \mathbf{X}\exists\theta' : \theta'(\theta(f_n^D))) \vee (M,\pi \models viol(n^\theta) \wedge \mathbf{X}\exists\theta' : \theta'(\theta(f_n^R)))$

$M,s \models \mathbf{X}failed(n^\theta)$ *iff* $M,s \models viol(n^\theta) \wedge \mathbf{X}timeout$

□

The first says that the norm remains in *active* status until there is no instance of deactivation condition occurring. The second says that the norm moves from the *active* to the *viol* state if there is no instance of the maintenance condition. The third says that the norm moves from the *active* to the *deactivated* state if there is an instance of the deactivation condition occurring and that the norm moves from the *viol* to the *deactivated* state if there is an instance of the repair condition occurring. The last says that the norm moves from the *viol* to the *failed* state if timeout occurs.

6.4.5 From abstract norm to norm instances

Now we have the apparatus needed to connect the fulfilment of an abstract norm and the fulfilment of its instances, and give semantic meaning to the operator proposed in Definition 10:

Definition 16 (Fulfilment of a norm based on the fulfilment of its instances)

$$M, \pi \models O_{f_n^R \leq timeout}([\alpha\ stit : f_n^M] \preceq f_n^D \mid f_n^A) \equiv_{def}$$
$$\exists \theta : M, \pi \models \mathbf{F}(\theta(f_n^A)) \Leftrightarrow M, \pi \models \mathcal{O}_{\theta(f_n^R) \leq timeout}([\alpha\ stit : \theta(f_n^M)] \preceq \theta(f_n^D) \mid \theta(f_n^A))$$

□

Informally: *the abstract norm is fulfilled if, and only if, for each possible instantiation of f_n^A through time, the obligations of the norm instances activated by f_n^A are fulfilled.*

6.4.6 Limitations and implications from a logic perspective

The formal semantics shown in this section are focused on binding deontic logics to our norms language for distributed systems. Special care has been taken on providing the capability of explicitly dealing with norm instances, and Standard Deontic Logics were not suited for that, and that is the reason of the apparent added complexity on the formalisation.

However, in order to claim that our formalisation is deontic in nature, we need a reduction from our semantics to deontic logics. This reduction is provided in the form of proofs in Appendix A. The conclusions we can get from this reduction is that for maintenance obligations, the translation to Standard Deontic Logics is complete. However, this is not the case for achievement obligations, in which the D axiom (the obligation of an action/state entails the non-obligation of the negation of such action/state) cannot be proven. This is an issue related to what the negation of an achievement really means from a semantic perspective. As discussed in Section A.1, while this is a problem from a formal perspective, in practice it should not affect the outcome.

On the other hand, the reduction to Dyadic Deontic Logic is complete, except for an open issue with the axiom $K2$ for which some debate is possible. For more details, we start such discussion in Section A.3.

Another limitation is the norm instance lifecycle we use for the formalisation and described in Section 6.3. This model is clearly inferior to the one presented in Section 2.2.1 in terms of flexibility. The motivation of this decision was based on pragmatism: these formal semantics were built with both monitoring and planning as targets. In order to be able to reduce the semantics to planning problems, LTL was the right logic to do so.

Therefore, this is a drawback to be taken into account when analysing the semantics of our norms. However, we consider that the results, from a practical perspective – being able to target PDDL opens up much of the research on automated planning for our framework – outweight this limitation.

Finally, another important issue to note is the fact that, in order to build the formalisation for our norm instance lifecycle, we have used a combination of LTL and first-order logic (FOL)[10]. This has been necessary due to the need for reasoning at the first-order level both about path states (e.g. a norm is never activated if there does not exist a state in which its activating condition holds true), and about substitution instances in order to enable the distinction between norms and norm instances (e.g. an activating condition may hold for a particular agent, but that does not mean that a norm instance has to be created to the rest of the agents of the system).

The use of first-order logic operators apparently rises the complexity of our formalism, as 1) (most, if not all) deontic logics are, in origin, merely based on modal logics with propositional content; and 2) first-order logic is not decidable. Regarding 1), we have tried to stay as close to deontic logics as possible with regards to the content or object of the norm by constraining the scope of its conditions to a propositional logic-based language (as explicitly stated in 6.4.1). With respect to 2), it is important to reiterate our intent on achieving a practical operationalisation and, therefore, decidability is not a concern as long as we can reduce our formalism, in a sound way, to a language close to the operational (and implementation) level. This will be the focus of Chapter 7.

6.5 Conclusions

In this chapter we have seen two separate formalisations: one for constitutive rules and the other for regulative rules. The rationale behind this separation is that both types of rules (norms) are totally different in nature: the nature of counts-as rules is ontological [Grossi 2007] while regulative rules are of deontic nature, with added complexity if we are to properly include norm instances.

The combination of both kinds of rules – constitutive, regulative – effectively gives us a formal framework to interpret and understand social reality in the sense of [Searle 2009]. In the next chapter, we will see how both formalisations can be reduced in a practical way into a normative monitor, in order to be used in simple computational

[10]A more exhaustive formalisation of the logic encompassing both, FO-LTL, can be found at [Kröger and Merz 2008].

components by using production systems as the implementation framework. Therefore, this will enable the implementation of actual components – including but not excluding both institutional (e.g. manager or police) and individual agents – capable of transforming external events into social reality.

CHAPTER 7

Normative monitor

‘Never ask an AI system what to do. Ask it to tell you the consequences of the different things you might do.’

The Robot and the Baby
JOHN MCCARTHY

In the previous chapter we have presented our formalisation for the inference of social reality (constitutive and regulative norms), one which includes norm instantiation explicitly in its formal semantics. In this chapter we will present a reduction of those semantics into production systems, and show how this reduction is implemented in a rule engine such as DROOLS.

7.1 NORMATIVE MONITOR

From the definitions introduced in Section 6.4.1, a *Normative Monitor* will be composed of the institutional specification, including norms, the current state of the world, and the current normative state.

In order to achieve an initial set rules, we need to establish a grounding for our formalism. First of all, we will define the lifecycle of a norm instance according to the LTL formalisations of the previous chapter. We will show how to transform the paths into transition rules, translating the principles of change in normative states into transition rules, effectively reducing our formalisation to a rule-based operational semantics.

In order to track the normative state of an institution at any given point of time, we assume the existence of a knowledge base, in which we will define four sets representing each of the lifecycle states: an active set AS, a violated set VS, a deactivated set DS, and a failed set FS, each of them containing norm instances in the form of tuples: $\{\langle n_i, \theta_j \rangle, \langle n_{i'}, \theta_{j'} \rangle, ..., \langle n_{i''}, \theta_{j''} \rangle\}$.

Definition 17 (Normative Monitor) *A Normative Monitor M_N for a set of norms N is a tuple*
$M_N = \langle N, AS, VS, DS, FS, S \rangle$*, where:*

- *s is the current state of the world, which corresponds to the current path state.*
- *N is the set of norms,*
- *$n \in N$, $\langle n, \theta \rangle \in AS \Leftrightarrow M, s \models active(n^\theta)$*
- *$n \in N$, $\langle n, \theta \rangle \in VS \Leftrightarrow M, s \models viol(n^\theta)$*
- *$n \in N$, $\langle n, \theta \rangle \in DS \Leftrightarrow M, s \models deactivated(n^\theta)$*
- *$n \in N$, $\langle n, \theta \rangle \in FS \Leftrightarrow M, s \models failed(n^\theta)$*

□

We denote Γ_{M_N} as the set of all possible configurations of a Normative Monitor M_N.

Definition 18 (Regulative transition rules) *The* Transition System TS_{M_N} *for a Normative Monitor M_N is defined by $TS_{M_N} = \langle \Gamma_{M_N}, \rhd \rangle$ where*

- *$\rhd$ is a* transition relation *such that $\rhd \subseteq \Gamma_{M_N} \times \Gamma_{M_N}$*

□

The inference rules for the transition relation $\rhd$ are described in Figure 7.1, where s_i stands for the current state and as, vs, ds, fs correspond to instances of the AS, VS, DS, FS sets of the Normative Monitor tuple.

However, the definition above does not take into account the dynamic aspects of incoming events affecting the state of the world through time. To extend our model we will assume that there is a continuous, sequential stream of events received by the monitor:

Definition 19 (Event) *An event e is a tuple $e = \langle \alpha, p \rangle$, where*

- *$\alpha \in P$[1], and*
- *$p \in S$ and is fully grounded.*

□

We define E as the set of all possible events, $E = \mathcal{P}(P \times S)$. Additionally, we can also add at this point the semantics for dynamic tracking counts-as rules activation and detection as defined in Section 6.2.1:

Definition 20 (Regulative and constitutive transition rules) *The* Labelled Transition System *for a Normative Monitor M_I* with constitutive and input capabilities *is defined by $\langle \Gamma, E, \rhd \rangle$ where*

- *E is the set of all possible events $e = \langle \alpha, p \rangle$*

[1] α is considered to be the asserter of the event. Although we are not going to use this element in this document, its use may be of importance when extending or updating this model.

Norm instance activated:

$$\frac{\theta(f_n^A) \vee \langle n, \theta\rangle \in as \qquad \neg\theta(f_n^D)}{M_N \rhd \langle N, as \cup \{\langle n, \theta\rangle\}, vs, ds, fs, s_{i+1}\rangle} \tag{7.1}$$

Norm instance violated:

$$\frac{\langle n, \theta\rangle \in as \qquad \neg\theta(f_n^M)}{M_N \rhd \langle N, as - \{\langle n, \theta\rangle\}, vs \cup \{\langle n, \theta\rangle\}, ds, fs, s_{i+1}\rangle} \tag{7.2}$$

Norm instance deactivated by fulfilment:

$$\frac{\langle n, \theta\rangle \in as \qquad \theta(f_n^D)}{M_N \rhd \langle N, as - \{\langle n, \theta\rangle\}, vs, ds \cup \{\langle n, \theta\rangle\}, fs, s_{i+1}\rangle} \tag{7.3}$$

Norm instance deactivated by reparation:

$$\frac{\langle n, \theta\rangle \in vs \qquad \theta(f_n^R)}{M_N \rhd \langle N, as, vs - \{\langle n, \theta\rangle\}, ds \cup \{\langle n, \theta\rangle\}, fs, s_{i+1}\rangle} \tag{7.4}$$

Norm instance failed:

$$\frac{\langle n, \theta\rangle \in vs \qquad timeout}{M_N \rhd \langle N, as, vs - \{\langle n, \theta\rangle\}, ds, fs \cup \{\langle n, \theta\rangle\}, s_{i+1}\rangle} \tag{7.5}$$

For all cases, $n \in N \wedge n = \langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout\rangle \wedge \theta \subseteq s_i$ is also part of the transition condition.

Figure 7.1: *Inference rules for the transition relation* $\rhd$

❧

- $\rhd$ *is a* transition relation *such that* $\rhd \subseteq \Gamma \times E \times \Gamma$.

□

The inference rules for the transition relation $\rhd$ are depicted in Figure 7.2. It is important to note that this set of transition rules are totally expanded from the definitions, in order to make the grounding to production systems in the next section more obvious. For the same reason, in this case we have assigned labels to each transition in an explicative way: ni for norm instantiation, nv for norm instance violation, and so on.

7.2 Formal reduction to production systems

In our approach, practical normative reasoning is based on a production system with an initial set of rules implementing the operational semantics described in Chapter 6. Production systems are composed of a set of rules, a working memory, and a rule interpreter or engine [Davis and King 1975]. Rules are simple conditional statements, usually of the form *IF a THEN b*, where *a* is usually called left-hand side (*LHS*) and *b* is usually called right-hand side (*RHS*). Our basic idea is that an agent can configure the production system by adding abstract organisational specifications and sets of counts-as rules.

Event processed:

$$\frac{e_i = \langle \alpha, p \rangle}{\langle\langle\langle i, s, as, vs, ds, fs\rangle, e_i\rangle, e_{i+1}\rangle \rhd_e \langle\langle i, s \cup \{p\}, as, vs, ds, fs\rangle, e_{i+1}\rangle} \tag{7.6}$$

Counts-as rule activation:

$$\frac{\exists \Theta, \exists f \in F(s), \exists \langle \gamma_1, \gamma_2, s_i\rangle \in C, s_i \subseteq s \wedge \Theta(\gamma_1) \equiv f \wedge \Theta(\gamma_2) \notin s}{\langle\langle\langle N, R, P, C, O\rangle, s, as, vs, ds, fs\rangle, e\rangle \rhd_{ca} \langle\langle\langle N, R, P, C, O\rangle, s \cup \{\Theta(\gamma_2)\}, as, vs, ds, fs\rangle, e\rangle} \tag{7.7}$$

Counts-as rule deactivation:

$$\frac{\exists \Theta, \exists f \in F(s), \exists \langle \gamma_1, \gamma_2, s_i\rangle \in C, s_i \not\subseteq s \wedge \Theta(\gamma_1) \equiv f \wedge \Theta(\gamma_2) \in s}{\langle\langle\langle N, R, P, C, O\rangle, s, as, vs, ds, fs\rangle, e\rangle \rhd_{ca} \langle\langle\langle N, R, P, C, O\rangle, s - \{\Theta(\gamma_2)\}, as, vs, ds, fs\rangle, e\rangle} \tag{7.8}$$

Norm instantiation:

$$\frac{\exists n = \langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout\rangle \in N \wedge \neg\exists n' \in N, \Theta\rangle \notin is \wedge \exists \Theta, \exists f' \in F(s), f' \equiv \Theta(f_n^A)}{\langle\langle\langle N, R, P, C, O\rangle, s, as, vs, ds, fs\rangle, e\rangle \rhd_{ni} \langle\langle\langle N, R, P, C, O\rangle, s, as \cup \{\langle n, \Theta\rangle\}, vs, ds, fs\rangle, e\rangle} \tag{7.9}$$

Norm instance violated:

$$\frac{\begin{array}{c}\exists n = \langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout\rangle \in N \wedge \langle n, \Theta'\rangle \in as \wedge \langle n, \Theta'\rangle \notin vs \wedge \\ \neg(\exists \Theta, \exists f' \in F(s), f' \equiv \Theta(f_n^M) \wedge \Theta \subseteq \Theta')\end{array}}{\begin{array}{c}\langle\langle\langle N, R, P, C, O\rangle, s, as, vs, ds, fs\rangle, e\rangle \rhd_{nv} \langle\langle\langle N, R, P, C, O\rangle, s, as - \{\langle n, \Theta'\rangle\}, \\ vs \cup \{\langle n, \Theta'\rangle\}, ds, fs\rangle, e\rangle\end{array}} \tag{7.10}$$

Norm instance deactivated by fulfilment:

$$\frac{\exists n = \langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout\rangle \in N \wedge \langle n, \Theta'\rangle \in as \wedge \exists \Theta, \exists f' \in F(s), f' \equiv \Theta(f_n^D) \wedge \Theta \subseteq \Theta'}{\langle\langle\langle N, R, P, C, O\rangle, s, as, vs, ds, fs\rangle, e\rangle \rhd_{nf} \langle\langle\langle N, R, P, C, O\rangle, s, as - \{\langle n, \Theta'\rangle\}, vs, ds \cup \langle n, fs, \Theta'\rangle\rangle, e\rangle} \tag{7.11}$$

Norm instance deactivated by reparation:

$$\frac{\exists n = \langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout\rangle \in N \wedge \langle n, \Theta\rangle \in vs \wedge \exists \Theta', \exists f \in F(s), f \equiv \Theta'(f_n^R) \wedge \Theta' \subseteq \Theta}{\langle\langle\langle N, R, P, C, O\rangle, s, as, vs, ds, fs\rangle, e\rangle \rhd_{nr} \langle\langle\langle N, R, P, C, O\rangle, s, as, vs - \{\langle n, \Theta\rangle\}, ds \cup \{\langle n, \Theta\rangle\}, fs\rangle, e\rangle} \tag{7.12}$$

Norm instance failed:

$$\frac{\exists n = \langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout\rangle \in N \wedge \langle n, \Theta\rangle \in vs \wedge \langle n, \Theta\rangle \notin fs \wedge timeout}{\langle\langle\langle N, R, P, C, O\rangle, s, as, vs, ds, fs\rangle, e\rangle \rhd_{nf} \langle\langle\langle N, R, P, C, O\rangle, s, as, vs - \{\langle n, \Theta\rangle\}, ds, fs \cup \{\langle n, \Theta\rangle\}\rangle, e\rangle} \tag{7.13}$$

Figure 7.2: *Inference rules for the transition relation* $\rhd$

❧

The implementation of rule-based norm operationalisation has already been explored in previous research [García-Camino, Noriega, and Rodríguez-Aguilar 2005; Paschke, Dietrich, and Kuhla 2005; Vázquez-Salceda and Alvarez-Napagao 2009], but these proposals are either not yet implemented in a real system or not using a high enough level of norm abstraction. Some recent approaches [García-Camino et al. 2009] define specific norm-oriented programming languages that treat norms as rules of a production system. However, such an approach requires for an special production system.

We solve this issue by combining the normative language presented in Section 5.1 with a reduction to a representation with clear operational semantics based on the framework in Section 6.4. This framework uses logic conditions that determine the state of a norm (active, fulfilled, violated). These conditions can be expressed in first-order logic and can be directly translated into *LHS* parts of rules, with no special adaptation needed. The implementation of the operational semantics in a production system to get a practical normative reasoner is thus straightforward. This allows agents for dynamically changing its organisational context at any moment, by *feeding* the production system with a new abstract organisational specification.

7.2.1 Reduction

In order to formalise our Normative Monitor as a production system, we will need to define several predicates to bind norms to their conditions: *activation, maintenance, deactivation,* and to represent normative state over norm instances: *violated, instantiated, failure,* and *fulfilled*. We will also use a predicate for the arrival of events: *event*. For the handling of the DNF clauses, we will use the predicates *holds* and *has_clause*.

Definition 21 (Production system predicates) *The set of predicates for our production system, for an institution $I = \langle N, R, P, C, O\rangle$, is:*

$$\begin{aligned}\mathcal{P}_I := O \cup \{&activated, maintained, deactivated,\\ &violated, instantiated, fulfilled, event, timeout, failed,\\ &holds, has\_clause, countsas\}\end{aligned}$$

□

The initial working memory $\mathcal{WM}_0$ should include the institutional specification in the form of the formulas included in the counts-as rules and the norms in order to represent the possible instantiations of the predicate $holds$, through the use of the predicate has_clause.

First of all, we need to have the bindings between the norms and their formulas available in the working memory. For each norm $n = \langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout\rangle$, these bindings will be:

$$\begin{aligned}\mathcal{WM}_n := \{&activation(n, f_n^A), maintenance(n, f_n^M),\\ &deactivation(n, f_n^D), timeout(n, timeout)\}\end{aligned}$$

As we assume the formulas from $wff(\mathcal{L}_O)$ to be in DNF form:

Definition 22 (DNF clause holding true) *We can interpret a formula as a set of conjunctive clauses $f = \{f_1, f_2, ..., f_N\}$, of which only one of these clauses f_i holding true is necessary for f holding true as well:*

$r^h := has\_clause(f, f') \wedge holds(f', \Theta) \Rightarrow \emptyset, \{holds(f, \Theta)\}$ □

For example, if $f = (p_1(x) \wedge p_2(y) \wedge ... \wedge p_i(z)) \vee ... \vee (q_1(w) \wedge q_2(x) \wedge ... \wedge q_j(y))$, then the initial facts to be in $\mathcal{WM}_0$ will be:

$\mathcal{WM}_0 := \bigcup_{f' \in f} has\_clause(f, f') = \{has\_clause(f, f_1), ..., has\_clause(f, f_2)\}$

Also, we have to include the set of repair norms by the use of the predicate $repair$, and the counts-as definitions by the use of the predicate $countsas$.

Definition 23 (Initial working memory) *The initial working memory $\mathcal{WM}_I$ for an institution $I = \langle N, R, P, C, O \rangle$ is:*

$$\begin{aligned}\mathcal{WM}_I := & \quad \bigcup^{n \in N}_{n \rightsquigarrow n'} repair(n, n') \quad \cup \\ & \bigcup_{n=\langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout \rangle \in N} (\mathcal{WM}_n \cup \mathcal{WM}_{f_n^A} \cup \mathcal{WM}_{f_n^M} \cup \mathcal{WM}_{f_n^D}) \quad \cup \\ & \bigcup_{c=\langle \gamma_1, \gamma_2, s \rangle \in C} (\{countsas(\gamma_1, \gamma_2, s)\} \cup \mathcal{WM}_{\gamma_1} \cup \mathcal{WM}_s)\end{aligned}$$

□

The rule for the detection of a holding formula is defined as $r_f^{hc} = \lceil f \rceil \Rightarrow \emptyset, \{holds(f, \sigma)\}$, where we denote as $\lceil f \rceil$ the propositional content of a formula $f \in wff(\mathcal{L}_O)$ which only uses predicates from O and the logical connectives $\neg$ and $\wedge$, and σ as the substitution set of the activation of the rule. Following the previous example:

$r_{f_1}^{hc} = p_1(x) \wedge p_2(y) \wedge ... \wedge p_i(z) \Rightarrow \emptyset, \{holds(f_1, \{x, y, z\})\}$

$r_{f_2}^{hc} = q_1(w) \wedge q_2(x) \wedge ... \wedge q_i(y) \Rightarrow \emptyset, \{holds(f_2, \{w, x, y\})\}$

Similarly as in Definition 23:

Definition 24 (Institutional reality) *The set of rules R_I^{hc} for detection of holding formulas for an institution $I = \langle N, R, P, C, O \rangle$ is:*

$R_I^{hc} := \bigcup_{n=\langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout \rangle \in N} (\bigcup_{f \in \{f_n^A, f_n^M, f_n^D\}} r_f^{hc}) \cup \bigcup_{c=\langle \gamma_1, \gamma_2, s \rangle \in C} (\bigcup_{f \in \gamma_1} r_f^{hc})$ □

By using the predicate $holds$ as defined above, we can translate the inference rules described at the beginning of this chapter. Please note that the rules are of the form $p, c \Rightarrow r, a$ as shown in Section 2.2.2. However, as we only need the c part to create a constraint proposition in the rules for norm instance violation and fulfillment, c is omitted except for these two particular cases.

Definition 25 (Translated rules for the transition relation[2])

Rule for event processing (7.6):

$r^e = event(\alpha, p) \Rightarrow \emptyset, \{\lceil p \rceil\}$

Rule for counts-as rule activation (7.7):

$r^{ca} = countsas(\gamma_1, \gamma_2, c) \wedge holds(\gamma_1, \Theta) \wedge holds(c, \Theta') \wedge \neg holds(\gamma_2, \Theta)$
$\Rightarrow \emptyset, \{\Theta(\lceil \gamma_2 \rceil)\}$

Rule for counts-as rule deactivation (7.8):

$r^{cd} = countsas(\gamma_1, \gamma_2, c) \wedge holds(\gamma_1, \Theta) \wedge \neg holds(c, \Theta') \wedge holds(\gamma_2, \Theta)$
$\Rightarrow \{\Theta(\lceil \gamma_2 \rceil)\}, \emptyset$

Rule for norm instantiation (7.9):

[2]See Figure 7.1.

$r^{ni} = activation(n, f) \wedge holds(f, \Theta) \wedge \neg instantiated(n, \Theta) \wedge \neg repair(n', n)$
$\Rightarrow \emptyset, \{instantiated(n, \Theta)\}$
Rule for norm instance violation (7.10):
$r^{nv} = instantiated(n, \Theta) \wedge maintenance(n, f) \wedge \neg holds(f, \Theta') \wedge repair(n, n'),$
$\forall \Theta', \Theta' \subseteq \Theta$
$\Rightarrow \{instantiated(n, \Theta)\}, \{violated(n, \Theta), instantiated(n', \Theta)\}$
Rule for norm instance fulfillment (7.11):
$r^{nd} = deactivation(n, f) \wedge instantiated(n, \Theta) \wedge subseteq(\Theta', \Theta) \wedge holds(f, \Theta'),$
$\Theta' \subseteq \Theta$
$\Rightarrow \{instantiated(n, \Theta)\}, \{fulfilled(n, \Theta)\}$
Rule for norm instance violation repaired (7.12):
$r^{nr} = violated(n, \Theta) \wedge repair(n, n')$
$\Rightarrow \{fulfilled(n, \Theta)\}, \{violated(n, \Theta)\}$
Rule for norm instance violation failed (7.13):
$r^{nr} = violated(n, \Theta) \wedge timeout(n)$
$\Rightarrow \{failed(n, \Theta)\}, \{violated(n, \Theta)\}$ □

Definition 26 (Set of institutional rules) *Following Definitions 22, 24 and 25, the set of rules for an institution $I = \langle N, R, P, C, O \rangle$ are:*
$\mathcal{R}_I := R_I^{hc} \cup \{r^h, r^e, r^{ca}, r^{cd}, r^{ni}, r^{nv}, r^{nd}, r^{nr}, r^{nf}\}$ □

Definition 27 (Institutional production system) *The production system $\mathcal{PS}_I$ for an institution I will be, from Definitions 21, 23 and 26:*
$\mathcal{PS}_I := \langle \mathcal{P}_I, \mathcal{WM}_I, \mathcal{R}_I \rangle$ □

7.3 Implementations

There are several production system implementations available, some of them widely used by the industry, such as Jess, Drools, Soar or Prova. In most of these systems rules are syntactically and semantically similar, so switching from one to the other would be quite simple. As production systems dynamically compile rules to efficient structures, they can be used as well to validate and verify the consistency of the norms.

A prototype of our normative reasoner has been implemented as a Drools program. Drools is an open-source Object-Oriented rule engine for declarative reasoning in Java [*JBoss Drools Business Rules, http://www.jboss.org/drools*], supported by the JBoss Community. Its rule engine is an implementation of the forward chaining inference Rete algorithm. Concepts are imported from standardised Description Logic OWL-DL ontologies into Java objects [Zimmermann 2009]. The use of Java objects inside the rule engine allows for an easier communication of concepts with the agent reasoning, the core of which is also implemented in Java.

In Drools we can represent facts by adding them to the knowledge base as objects of the *Predicate* class. Predicates are dynamically imported from standardised Description Logic OWL-DL ontologies into Java objects using the *OWL2Java* tool[Zimmermann 2009], as subclasses of a specifically designed *Predicate* class. The following shows an example of the insertion of $Mayor(a)$ into the knowledge base to express that a (represented as object $obj3$ of the domain) is in fact a mayor.

```
Object obj3 = new Object();
ksession.insert(obj3);
ksession.insert(new Mayor(obj3));
```

In DROOLS we can represent facts by adding them to the knowledge base as objects of the class *Predicate*. The following shows an example of the insertion of $Mayor(a)$ into the knowledge base to express that a (represented as object a of the domain) is in fact a mayor.

```
ksession.insert(new Mayor(a));
```

The *Predicate* class is designed specifically for our implementation and is the superclass of every predicate in the system. We use this abstraction as a basis to reason about norms with DROOLS.

7.3.1 Handling of constitutive contexts

We implement the concept of *Context* as a subclass of *Predicate*, asserting its instances into the knowledge base:

```
ksession.insert(Context.CAR_CRASH);
ksession.insert(Context.FLOODING);
ksession.insert(Context.UNIVERSAL);
```

Defining contexts as concepts in the knowledge base allows us to also refer to them explicitly and reason about them. This is an important advantage over implementations where contexts are mere labels on the counts-as relations between concepts.

In order to define the proper classificatory counts-as in a specific context, the predicate *ClassificatoryCountsAs* is introduced. This predicate allows for the expression of classificatory relations *between classes* with respect to a context.

```
ksession.insert(
new CountsAs(
  VerbalOK.class,
  Inform.class));
new CountsAs(
  PdaOK.class,
  Inform.class));
new CountsAs(
  PdaOK.class,
  TraceableInform.class));
```

Figure 7.3: *Definition of classificatory counts-as rules.*

❧

The expressions of Figure 7.3 show two examples of the classificatory counts-as, where the statements respectively describe that, in the *universal* context, a verbal OK *counts-as* an inform, and a PDA OK *counts as* both an inform and a traceable inform.

Figure 7.4, on the other side, shows the proper classificatory counts-as for the example. In this case, and following the example of crisis management presented in Section 6.1, each counts-as refers to different contexts: an inform *counts-as* a proper inform in a car crash scenario, but in a flooding scenario a traceable inform *counts-as* a proper inform.

```
ksession.insert(
new ClassificatoryCountsAs(
  Inform.class,
  ProperInform.class,
  Context.CAR_CRASH));
new ClassificatoryCountsAs(
  TraceableInform.class,
  ProperInform.class,
  Context.FLOODING));
```

Figure 7.4: *Definition of proper classificatory counts-as rules.*

❧

To implement the uniqueness criterium specified in Section 6.2.3, which allows for more efficient runtime use of the counts-as rules, we implemented translation into Drools rules to create internal parallel sets of contexts (based on the intuitions expressed in figures 6.1 and 6.2 in Section 6.2.3). The first rule of Figure 7.5 shows how this splitting is done, while the second rule of Figure 7.5 gives an example of how one can identify in which (original) context a counts-as rule was formulated.

Figure 7.6 then shows an example of the context splitting. From three counts-as rules, of which two of them are the same for two different contexts, the result will be two contexts.

The first rule of the example expresses that ambulances count as a means of evacuation in the context of grip-2; the second expresses that ambulances also count as a means of evacuation in the context of grip-3; the third rule expresses that in the context of grip-3 army trucks also count as a means of evacuation. After the splitting of contexts grip-2 and grip-3 containing just these three rules we end up with two contexts, namely the context which contains the rules that are present in both grip-2 and grip-3 (ambulances count as means of evacuation), and the context which gives the refinement of being in context grip-3, namely that army trucks also count as means of evacuation. The result being the `GRIP2GRIP3` context containing the rule about ambulances being evacuation means (now unique, as there is no need to specify it twice) and the `GRIP3` context containing only the rule specifying that army trucks are evacuation means. As explained in Section 6.2.3, this split allows for an easy and efficient means

```
rule "creation of running contexts"
  when
    ClassificatoryCountsAs(a : c1, b : c2)
    and
    lc : TreeSet() from collect(
      ClassificatoryCountsAs(c1 == a, c2 == b))
  then
    RunningContext rc;
    rc = new RunningContext(lc);
    insertLogical(rc);
end

rule "identify running contexts"
  when
    cca : ClassificatoryCountsAs(c : context)
    and
    rc : RunningContext(countsas contains cca)
  then
    insertLogical(
      new RunningContextIdentifier(rc, c));
end
```

Figure 7.5: *Context splitting.*

❧

to check the similarities and differences between the contexts GRIP-2 and GRIP-3[3].

The internal effect of a context activation is the activation of all its shared contexts (see Figure 7.7). With the contexts active, their counts-as rules will be instantiated as active counts-as rules in the rule engine. The counts-as rules are fired whenever there is a matching predicate. The effect of a fired counts-as rule is that for each instance of the first predicate of the rule, a new instance of the second predicate of the rule is created.

Closure is provided in the monitor by automatically detecting which context should be active based on the active counts-as rules. Figure 7.8 shows the rules implemented for this purpose. The first rule detects if all the proper classificatory counts-as rules for a certain shared context are instantiated, in which case that shared context will be activated automatically. The second rule checks if all the shared contexts that belong to a user defined context are active, in which case the context will be activated.

By using these rules we can identify the concept of a context (like GRIP-2) with the counts-as rules related to that context. Having this constitutive relation between a context and the counts-as rules available we can now also handle the following scenario of the crisis management.

[3]Note that in this example GRIP-3 ended up as a subcontext of GRIP-2 because of the limited scope of the example. In reality, there are other differences between these contexts which would show that they instead overlap.

```
ksession.insert(
  new ClassificatoryCountsAs(
    Ambulance.class,
    MeansOfEvacuation.class,
    Context.GRIP2));
ksession.insert(
  new ClassificatoryCountsAs(
    Ambulance.class,
    MeansOfEvacuation.class,
    Context.GRIP3));
ksession.insert(
  new ClassificatoryCountsAs(
    ArmyTruck.class,
    MeansOfEvacuation.class,
    Context.GRIP3));

(after kession.fireAllRules())

[GRIP2GRIP3, GRIP3]
```

Figure 7.6: *Example of context splitting.*

❧

Suppose the hospital has to be evacuated due to a flooding. There are not enough ambulances available to evacuate all people in time. The commander (chief medic at the location) checks to see what can be done. He can use (special) army trucks. However, army trucks do not (in general) count-as ambulances. The commander can check (with the Drools implementation) that army trucks count-as ambulances in the context of GRIP-3. (They are part of constituting GRIP-3). So, the commander decides to move to the context of GRIP-3. Now he has to check what other rules constitute GRIP-3. One of them states that in GRIP-3 the mayor counts-as commander. This means that the commander has to transfer his command to the mayor. Moreover, in a flooding scenario, as stated previously, only a traceable inform *counts-as* an appropiate inform. That means that all agents should be aware of the new context and act accordingly, being forced to adapt to a traceable informing mechanism if they were not using it.

The scenario shows that we need the context as an explicit concept and also we need the constitutive aspect of the counts-as rules that define the context in order for the commander to be able to define a switch to another context (GRIP level) and realizing the consequences of this switch. The Drools implementation presented above enables us to do this.

7.3.2 Monitoring of regulative norms

Drools programs can be initialised with a rule definition file. However, its working memory and rule base can be modified at run-time by the Java process that is running the rule engine. We take advantage of this by keeping a fixed base, which is a file with

```
rule "activate running contexts"
  when
    ContextActive(c : context)
    and
    RunningContextIdentifier(
      rc : runningContext, context == c)
  then
    insertLogical(new RunningContextActive(rc));
end

rule "classificatory counts-as"
  when
    rc : RunningContextActive(
      ca : ClassificatoryCountsAs(
        y1 : c1, y2 : c2))
    and
    obj : Predicate(class == y1)
  then
    insertLogical(new CountsAs(y1, y2));
end

rule "counts-as"
  when
    c : CountsAs(y1 : c1, y2 : c2)
    and
    obj : Predicate(class == y1)
  then
    Predicate instance;

    instance = (Predicate)(
      ((Class)y2).newInstance());
    instance.setObject(obj.getObject());
    insertLogical(instance);
end
```

Figure 7.7: *Activation of counts-as rules.*

❧

fixed contents implementing the rules from Definition 22 and 25, which are independent of the institution, and having a parser for institutional definitions that will feed the rules from Definition 24, which are dependent on the institution (see Figure 7.12). The institutional definitions we currently use are based on an extension of the XML language presented in Section 5.2.1.

The base rules (see Definitions 22 and 25) have been quite straightforward and the translation is almost literal. The contents of the reusable DROOLS file is shown in Figure 7.9. The last rule of the Figure is the declarative implementation of the predicate *SubsetEQ* to represent the comparison of substitutions instances $\Theta \subseteq \Theta'$, needed for the cases of norm instance violation and fulfillment. In our implementation in *Drools*, substitution instances are implemented as *Set*<*Value*> objects, where *Value* is a tuple

```
rule "activate running context"
  when
    rc : RunningContext(cal : countsas)
    and
    forall(
      ca : ClassificatoryCountsAs(a : c1, b : c2)
      from cal
        CountsAs(c1 == a, c2 == b))
  then
    insertLogical(new RunningContextActive(rc));
end

rule "activate context by its running contexts"
  when
    c : Context()
    and
    forall(
      RunningContextIdentifier(
      rc : runningContext, context == c)
        RunningContextActive(runningContext == rc)
    )
  then
    insertLogical(new ContextActive(c));
end
```

Figure 7.8: *Automatic activation of contexts.*

❧

⟨*String, Object*⟩.

The rest of the rules (see Definitions 24) are automatically generated from the institutional specifications and inserted into the Drools rule engine. An example of two generated rules for the traffic scenario is shown in Figure 7.10.

The initial working memory is also automatically generated by inserting objects (facts) into the Drools knowledge base following Definition 23. An example for the traffic scenario is also shown in Figure 7.11. Please note that this is not an output of the parser, but a representation of what it would execute at run-time. The resulting architecture is depicted in Figure 7.12.

7.4 Conclusions

In this chapter we have shown a concrete implementation of abstract norms that can be used by agents to reason about social reality. One of the key issues of connecting norms to concrete actions to be taken is that we need (at least) two uses of the counts-as relation. One to connect brute facts (or events) to institutional states and actions.

```
rule "holds"
when
  HasClause(f : formula, f2 : clause)
  Holds(formula == f2, theta : substitution)
then
  insertLogical(new Holds(f, theta));
end

rule "event processed"
when
  Event(a : asserter, p : content)
then
  insertLogical(p);
end

rule "counts-as activation"
when
  CountsAs(g1 : gamma1, g2 : gamma2, s : context)
  Holds(formula == g1, theta : substitution)
  Holds(formula == s, theta2 : substitution)
  not Holds(formula == g2, substitution == theta)
then
  Formula f;

  f = g2.substitute(theta);
  insert(f);
end

rule "counts-as deactivation"
when
  CountsAs(g1 : gamma1, g2 : gamma2, s : context)
  Holds(formula == g1, theta : substitution)
  not Holds(formula == s, theta2 : substitution)
  Holds(formula == g2, substitution == theta)
  f : Formula(content == g2, grounding == theta)
then
  retract(f);
end

rule "norm instantiation"
when
  Activation(n : norm, f : formula)
  Holds(formula == f, theta : substitution)
  not Instantiated(norm == n, substitution == theta)
  not Repair(n2 : norm, repairNorm == n)
then
  insert(new Instantiated(n, theta));
end

rule "norm instance violation"
when
  ni : Instantiated(n : norm, theta : substitution)
  Maintenance(norm == n, f : formula)
  not (SubsetEQ(theta2 : subset, superset == theta)
  and Holds(formula == f, substitution == theta2))
  Repair(norm == n, n2 : repairNorm)
then
  retract(ni);
  insert(new Violated(n, theta));
  insert(new Instantiated(n2, theta));
end

rule "norm instance fulfillment"
when
  Deactivation(n : norm, f : formula)
  ni : Instantiated(norm == n, theta : substitution)
  SubsetEQ(theta2 : subset, superset == theta)
  Holds(formula == f, substitution == theta2)
then
  retract(ni);
  insert(new Fulfilled(n, theta));
end

rule "norm instance violation repaired"
when
  ni : Violated(n : norm, theta : substitution)
  Repair(norm == n, n2 : repairNorm)
  Fulfilled(norm == n2, substitution == theta)
then
  retract(ni);
end

rule "subseteq"
when
  Holds(f : formula, theta : substitution)
  Holds(f2 : formula, theta2 : substitution)
  eval(theta.containsAll(theta2))
then
  insertLogical(new SubsetEQ(theta2, theta));
end
```

Figure 7.9: *Translation of base rules to Drools*

❧

```
rule "N1_activation_1"
when
  n : Norm(id == "N1")
  Activation(norm == n, f : formula)
  Enacts(X : p0, p1 == "Driver")
  IsDriving(p0 == X)
then
  Set<Value> theta = new Set<Value>();
  theta.add(new Value("X", X));
  insert(new Holds(f.getClause(0), theta));
end

rule "C1_1"
when
  c : CountsAs(g1 : gamma1)
  Exceeds(D : p0, 50 : p1)
then
  Set<Value> theta = new Set<Value>();
  theta.add(new Value("D", D));
  insert(new Holds(g1.getClause(0), theta));
end
```

Figure 7.10: *Rules for the traffic scenario*

❧

```
ksession.insert(norm1);
ksession.insert(norm2);
ksession.insert(new Repair(norm1, norm2));
ksession.insert(new Activation(norm1, fn1a));
ksession.insert(new Maintenance(norm1, fn1m));
ksession.insert(new Deactivation(norm1, fn1d));
ksession.insert(new HasClause(fn1a, fn1a1));
ksession.insert(new HasClause(fn1m, fn1m1));
ksession.insert(new HasClause(fn1d, fn1d1));
/*        ...same for norm2...        */
ksession.insert(new CountsAs(c1g1, c1g2, c1s));
ksession.insert(new HasClause(c1g1, c1g11));
ksession.insert(new HasClause(c1g2, c1g21));
ksession.insert(new HasClause(c1s, c1s1));
```

Figure 7.11: *Facts for the traffic scenario*

❧

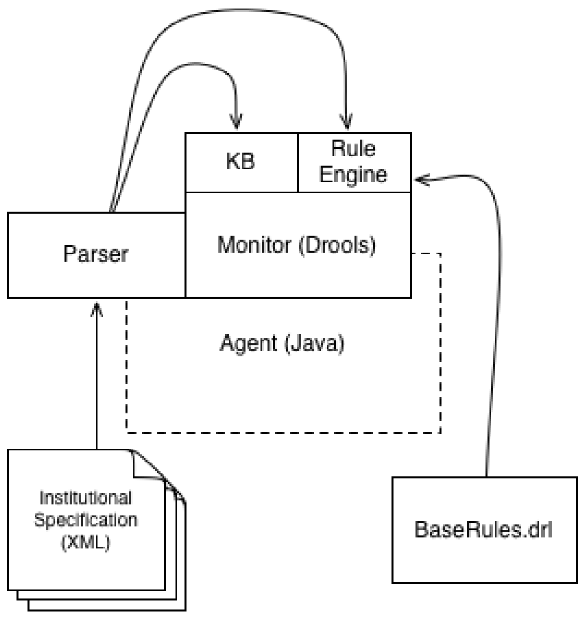


Figure 7.12: *Architecture of the Drools implementation*

❧

The second one to connect these institutional facts and actions to their normative interpretation.

We have also shown that the context of the norms and the counts-as relation is important in the type of applications that we are using this framework for. Thus we cannot suffice with a pure classificatory implementation of the counts-as relation, which would be straightforward. Instead we have to use an implementation that takes the context of the norms and the counts-as relation into account. One of the consequences of using this proper classificatory version of the counts-as relation is that it is no longer transitive. We have to check whether contexts change between the rules in order to know whether they can be combined.

We have shown how the above requirements have been met by the implementation in DROOLS. In DROOLS we can explicitly connect the context to the counts-as rules and use it as a constraint on its use. Agents can make use of the DROOLS engine to reason on what specific course of action they should pursue in order to comply to the norms that are imposed by the context they are in. However, they can do more than just that. They can also reason about the consequences of changing the context. In the example, the commander has to the ability to change the context by informing the mayor about the disaster. Of course this action of the commander itself can be regulated by norms that state that the mayor can only be involved if the disaster gets too big (according to some criteria).

The resulting semantic framework presented in Part II directly tackles at the same time three important problems related to the practical materialization of norm-aware systems: straightforward connection between the deontic level and the operational semantics, the formalization of explicit norm instances, and the unambiguity of semantic interpretation across implementation domains. We have done so by building, upon diverse previous work, a conection between deontic statements and temporal logics, and between temporal logics to fluents and transition rules. Previous work also shows [Alvarez-Napagao et al. 2011; Panagiotidi and Vázquez-Salceda 2011] that from the latter representations the translation to the implementation level is also clear.

The implementation of rule-based norm operationalisation has already been explored in previous research. Some approaches [Paschke, Dietrich, and Kuhla 2005; Strano, Molina-Jimenez, and Shrivastava 2008] directly define the operationalisation of the norms as rules of a specific language, not allowing enough abstraction to define norms at a high level to be operationalised in different rule engine specifications. [García-Camino, Noriega, and Rodríguez-Aguilar 2005] introduces a translation scheme, but it is bound to Jess by using specific constructs of this language and it does not support constitutive norms. Other approaches like [García-Camino et al. 2009] define rule-based languages with expressive constructs to model norms, but they are bound to a proper interpreter and have no grounding on a general production system, requiring the use of an intentionally crafted or modified rule engine. For example, in [Governatori 2005; Hübner, Boissier, and Bordini 2009], obligations, permissions and prohibitions are asserted as facts by the execution of the rules, but the actual monitoring is out of the base rule engine used.

[Tinnemeier, Dastani, and Meyer 2009] introduces a language for defining an organisation in terms of roles, norms, and sanctions. This language is presented along with an operational semantics based on transition rules, thus making its adoption by a general production system straightforward. Although a combination of counts-as rules and sanctions is used in this language, it is not expressive enough to support regulative norms with conditional deontic statements.

We solve these issues by combining a lightweight, expressive normative language (see Section 5.1) with a reduction to a representation with clear operational semantics for deontic norms and the use of counts-as rules for constitutive norms (see Chapter 6). The formalism presented in this thesis uses logic conditions that determine the state of a norm (active, fulfilled, violated). These conditions can be expressed in propositional logic at the moment and can be directly translated into *LHS* parts of rules, with no special adaptation needed. The implementation of the operational semantics in a production system to get a practical normative reasoner is thus straightforward. This allows agents for dynamically changing its institutional context at any moment, by *feeding* the production system with a new abstract institutional specification.

Our intention is not to design a general purpose reasoner for normative agents, but a practical reasoner for detecting event-driven normative states. This practical reasoner can then be used as a component not only by normative agents, but also by monitors or managers. Normative agents should deal with issues such as planning and future possibilities, but monitors are focused on past events. For such a practical reasoner, the expressivity of actions languages like $C+$ is not needed, and a simple yet efficient solution is to use production systems, as opposed to approaches more directly related to offline verification or model checking, such as [Kyas, Prisacariu, and Schneider 2008].

Mere syntactical translations are usually misleading in the sense that rule language specific constructs are commonly used, constraining reusability [García-Camino, Noriega, and Rodríguez-Aguilar 2005; Governatori 2005; Paschke, Dietrich, and Kuhla 2005]. However, as we have presented in this chapter a reduction to a general version of production system semantics, any rule engine could fit our purposes. This effectively allows our grounding to be applied to a wide range of languages and deployment platforms through libraries that can compile definitions in our norm language into simple rules in a very efficient way. As opposed to [Governatori 2005; Hübner, Boissier, and Bordini 2009], our reduction ensures that the whole monitoring process is carried out entirely by a general production system, thus effectively decoupling normative state detection and agent reasoning.

Drools is an open-source powerful suite supported by JBoss, the community, and the industry, and at the same time it is lightweight enough while including key features that we are or will be using in future work. As an advantage over other alternatives, it includes features relevant to our topic, e.g. event processing, workflow integration. Its OO approach makes it easy to be integrated with imperative code (Java), and OWL-DL native support is expected in a short time. The monitoring system is available at `http://sf.net/projects/ict-alive` under a GPL license.

PART III

WRAP-UP

CHAPTER 8

From theory to practice

But once a law has been swiftly made, it has a constant and lasting force and needs to be enforced all the time, or at least there must always be someone on duty to enforce it when there is need for that. So there must be a power that -— unlike the legislature -— is always in existence, a power that will see to the enforcement of the laws that have been made and not repealed. That is how the legislative and executive powers come to be separated in many commonwealths.

John Locke
SECOND TREATISE OF GOVERNMENT

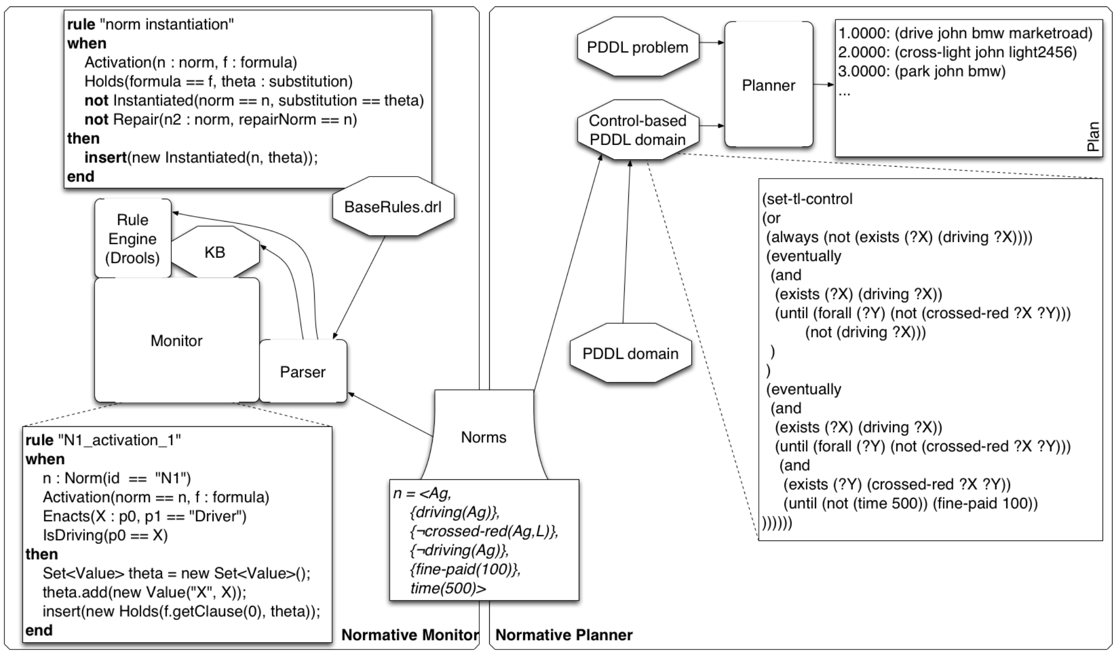
Along the whole Part II, we have explored several computational elements: a language for norms, its operational semantics, and a reduction to production systems translatable to rule engines. These elements are specially tailored for enabling the monitoring of social reality in distributed systems, in a broad sense of distributed systems: multi-agent systems and service-oriented architectures.

All these elements have roots on formalisms while being targetted at practical systems. An example of that is the fact that at the end of Chapter 7 we present implementations that ground our work. Because this grounding is done on rules of a production system, they can be applied to a very wide range of programming languages and therefore are suitable for integration not only in agents but in services.

However, does this mean that we can automatically achieve an adequate level of social order (see Section 3.2)? What about enforcement? In this chapter we try to tackle these issues by showing: 1) hints on how our formalism for monitoring can be generalised to other normative-related tasks, and 2) a proposal for an architecture for SOA governance that fits our computational elements.

8.1 Generalising our approach

The focus of this thesis is on monitoring of norms, and this work was first published in [Alvarez-Napagao et al. 2011]. However, this operationalisation evolved after merging this work with that of Panagiotidi *et. al* [Panagiotidi and Vázquez-Salceda 2011] about planning in norm-aware rational agents. As a result of this joint work, the operational semantics were generalised in a way that allowed, in the first place, to maintain the same reduction to production systems, and in the second place and no less importantly, to enable a new reduction to fluent-based semantics, and the subsequent translation to implementations in PDDL. An example can be seen in Figure 8.1.



```
rule "norm instantiation"
when
    Activation(n : norm, f : formula)
    Holds(formula == f, theta : substitution)
    not Instantiated(norm == n, substitution == theta)
    not Repair(n2 : norm, repairNorm == n)
then
    insert(new Instantiated(n, theta));
end
```

```
rule "N1_activation_1"
when
    n : Norm(id  ==  "N1")
    Activation(norm == n, f : formula)
    Enacts(X : p0, p1 == "Driver")
    IsDriving(p0 == X)
then
    Set<Value> theta = new Set<Value>();
    theta.add(new Value("X", X));
    insert(new Holds(f.getClause(0), theta));
end
```

n = <Ag,
{driving(Ag)},
{¬crossed-red(Ag,L)},
{¬driving(Ag)},
{fine-paid(100)},
time(500)>

```
1.0000: (drive john bmw marketroad)
2.0000: (cross-light john light2456)
3.0000: (park john bmw)
...
```

```
(set-tl-control
(or
 (always (not (exists (?X) (driving ?X))))
 (eventually
  (and
   (exists (?X) (driving ?X))
   (until (forall (?Y) (not (crossed-red ?X ?Y)))
        (not (driving ?X)))
  )
 )
 (eventually
  (and
   (exists (?X) (driving ?X))
   (until (forall (?Y) (not (crossed-red ?X ?Y)))
    (and
     (exists (?Y) (crossed-red ?X ?Y))
     (until (not (time 500)) (fine-paid 100))
))))))
```

Figure 8.1: *Example of reductions to monitoring and planning*

❧

Because both reductions have the same formal grounding (see Section 6.4), the semantics of monitoring are equivalent to the semantics of planning. Additionally, because the operational semantics have valid reductions to deontic logic (see Appendix A), we can consider that this generalisation is valid from a formal perspective.

This is a novel contribution that results from our work, allowing us to:

1. Have a more compact semantics for both (planning and monitoring) domains,
2. have the capability to ensure that the same semantics are understood between both domains (i.e., that the agents' norm interpretation is aligned with the compliance mechanism one), and

3. be able to express norms in deontic terms at a high level and translate them automatically to both PDDL and production systems.

Solving the planning problem is not related to enforcement but to practical reasoning of norm-aware agents. However, both problems solved by our reductions – monitoring and planning – cover much of the complexity that real-time systems commonly need. Therefore, we propose that it is possible to apply our operational semantics to the field of enforcement, either by applying combinations of monitoring and planning at the institutional or administrative level, or by creating new reductions to solve rule-style or state-space search-based enforcement techniques.

8.2 Advancing towards SOA governance

In this section we introduce our proposal for a generic SOA governance architecture based on norms. Although the current version is mainly designed for service-oriented architectures, it can be easily adapted to be used also by agents in an agent platform. The global picture of this architecture is shown in Figure 8.3.

8.2.1 Use case: organ transplant management

As an example, we will use an organ transplant distributed management application, a result of the EU-Provenance project[1]. The Organ Transplant Management Application (OTMA) is an Agent-Mediated Electronic Institution for the distribution of organs and tissues for transplantation purposes. It extends CARREL [Vázquez-Salceda et al. 2003], the aim of which was to help speeding up the allocation process of solid organs for transplantation to improve graft survival rates. As opposed to CARREL, OTMA uses standard web service technology and is able to interact with provenance stores in order to keep track of the distributed execution of the allocation process for auditing purposes.

Figure 8.2 summarises the different administrative domains (solid boxes) and units (dashed boxes) that are modeled in the OTMA system: the Organ Transplant Authority (OTA), a store of Electronic Healthcare Records (EHCR), and the management systems for the OTA recipient waiting lists (WL). Each of these interact with each other through agents (circles in the figure) that exchange information and requests through messages. In a transplant management scenario, one or more hospital units may be involved: the hospital transplant unit, one or several units that provide laboratory tests and the EHCR subsystem which manages the health care records for each institution. The diagram also shows some of the data stores that are involved: apart from the patient records, these include stores for the transplant units and the WL. Hospitals that are the origin of a donation also keep records of the donations performed, while hospitals that are recipients of the donation may include such information in the recipient's patient record. The OTA has also its own records of each donation, stored case by case.

[1]EU-Provenance was a STREP project (511085) funded by the 6th Framework Programme of the European Commission.

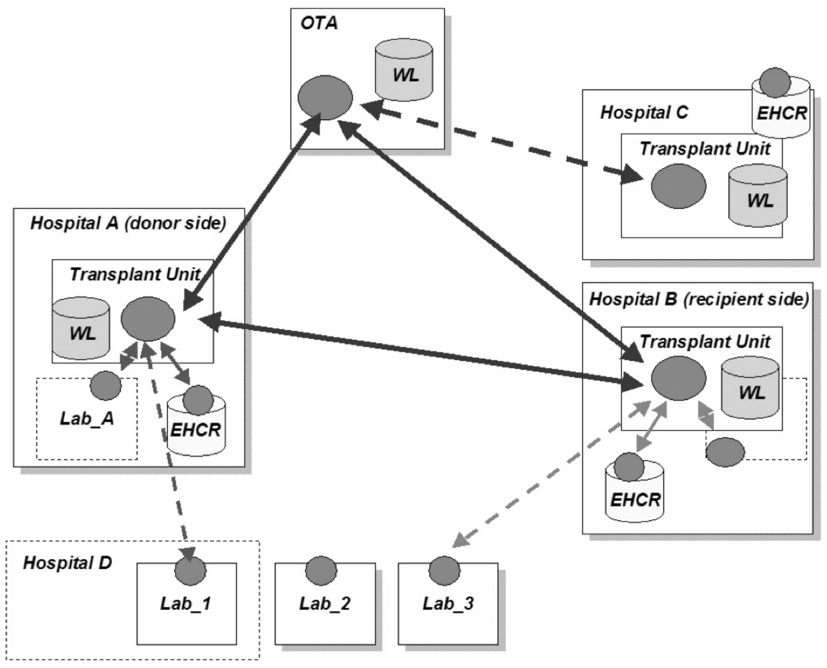


Figure 8.2: *Actors in the OTMA system. Each medical unit is represented by an agent (circle in figure).*

8.2.2 A generic SOA governance architecture based on norms

The examples shown in the form of rules are based on the Jess rule engine. The Jess language was the first target for the reductions of our normative monitor.

When application agents enter for the first time in the institution, they can access the norms and the ontological definitions in the context manager module. Agents log the relevant events by creating them and sending them to the observer agent, which is the one that keeps the event store that acts as a log for all the reported events and annotates the causal relationships between events (their provenance). The observer agent sends some of those reported events to one or more Enforcement agents (each of those should have previously registered the list of events they need to be notified of, according to the norms each of them has to enforce). Each enforcement agent combining the reported events with the norms that such agent is responsible to enforce. If a violation is detected, then the enforcement agent should see to it that the repair condition is met, as specified in the norms.

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

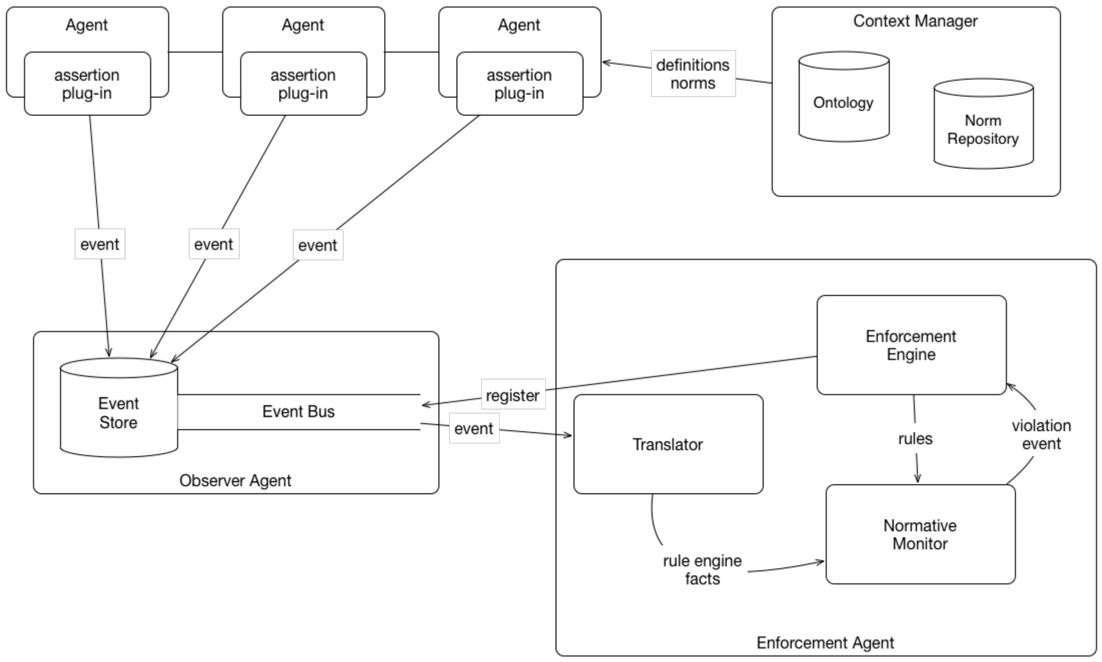


Figure 8.3: *Layout of the architecture*

❧

8.2.2.1 Context Manager

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

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

- a specific vocabulary defining the meaning of the terms used in the interactions between the agents of the institution,
- shared descriptions about resources, processes and/or 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 institution. Each norm complies with the language described in Section 5.2.

8.2.2.2 *Application Agent*

The Application Agents are those agents that interact within each other inside the institution and its context. They have the same generic role as any element of a distributed system – services or intelligent agents – and they do not necessarily have an active role in norm enforcement, but they should report all relevant events to the observer agent. These events will be used by the enforcement agents to enforce the norms applying to the application agents' behavior.

Before an Application Agent can start its activity within the institution, it has to retrieve the definitions and norms 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, if it has any. 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 (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 Event Store, ensuring a safe, reliable, and accurate recording of the events generated by the agents execution.

To avoid that the generation of events stops the execution of the agent or that some events get lost due to temporary unavailability communication problems between the Application Agent and the Observer Agent, the plug-in uses an event queue (such as RabbitMQ[2]), which allows the event submission to be completely asynchronous and loosely coupled to the core of the agent, avoiding critically blocks in its execution.

8.2.2.3 *Observer Agent*

An Observer Agent has the responsibility to safely register and maintain the environmental events and state changes of the 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 does not match with their needs or arrives too late.

[2] https://www.rabbitmq.com

The gathering is handled by the Event 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.

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

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

Event Store. The Event 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 institution which will make use of them. As a repository of raw events, 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 Event Bus, as sets of events.

Enforcement Agent. The Enforcement Agents are responsible for the fulfillment of a subset of the norms of the context in the 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. 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 templates for the events they are supposed to look after. Once registered, they will be properly notified in the form of event. Therefore, there is no need of a direct communication between an Enforcement Agent and the Application Agents. The Translator converts these events into a format understandable by the Enforcement Agent. Another component is needed for detecting the violations. In our case we are using our Normative Monitor (see Chapter 7), which matches the events, in the form of facts of the monitor rule engine, and the norms, in the forms of production 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 events to the Enforcement Agent when they are of any interest. However, the Normative Monitor is an instance of a rule engine. The Translator is a simple component which parses these events and generates facts of such rule engine.

```
(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 8.4: *An example of translation rule from p-assertion to Jess* `asserted` *fact*

❧

The Translator obtains the translation rules from the Context Manager. In Figure 8.4 we show one example of a rule that obtains a rule engine assertion of an organ assignment, taking an organ assignment event as input. This rule parses the formatted event, keeping only the relevant data for the system and generating an asserted fact, which will be added to the rule 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 event and written in a rule engine fact.

When an agent records an event indicating the confirmation of an assignment, it includes content compliant with the OTMA 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.

We have implemented an automatic translator of rules, capable of parsing an schema and generating one rule per each kind of event.

Normative Monitor. Once the Enforcement Engine has received the norms from the Context Manager, it creates a set of rule engine rules out of them and sends them to the Normative Monitor. This component is, in fact, an instance of a rule 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 Normative Monitor, this component needs to provide the rule engine with rules for each norm.

For instance, let us see how norm *N37: "The Organ Transplant Authority is obliged to ensure the compatibility of the organ with the recipient patient before doing the assignment of the organ to that patient"* is handled. A violation 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 modelled in Figure 8.5. The rule shown in the figure takes as input two facts: the fact generated (using the translation rule shown in Figure 8.4) 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 rule 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 8.5: *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 Event Bus.

Therefore, for norm *N37* all the measures should be executed by the Enforcement Agents, as they are all institutional.

8.2.3 Mapping our architecture to SOA governance

The rationale behind the design of this architecture is a combination between the elements that we have available from Part II and the architectural requirements for SOA governance summarised in Section 3.2.2. As a result of this, there are many of the components belonging to these requirements that can be identified as components of our architecture (see Table 8.1).

All the components are mapped one to one, e.g. the Normative Monitor is a perfect match with respect to the Rule Engine required in SOA governance. There is but one exception: the Registry. This component is too coupled with the actual underlying implementation of the individuals in the distributed system. This is usually already provided by the corresponding architecture, such as the Directory Facilitator in FIPA-compliant implementations.

SOA governance component	**Proposed mapped component**
Registry	–
Service	Agent
Repository	Context Manager
Policy enforcement points	Enforcement Engine
Rule engine	Normative Monitor
Environment	Enforcement Agent

Table 8.1: *Mapping between components required by SOA governance and components of our proposed architecture*

❧

At this point, our proposed architecture allows us to manage the Design-Time and Run-Time aspects of SOA governance. Change-Time governance requires an extension of our normative monitor to support dynamic promulgation and abrogation of norms, something that is already being researched by [Gómez-Sebastià and Alvarez-Napagao 2012].

8.3 Conclusions

This chapter has been dedicated to giving some insight on how to tackle some open issues that arise from trying to analyse the impact of the contributions presented in Part II in real-world scenarios: how to use our monitoring mechanism in systems that also need to cope with enforcement and, from a more general perspective, how to design an architecture based on our monitoring mechanism to achieve governance in distributed systems.

With respect to the former issue, we propose that the operational formalisation presented in Chapter 6 is highly likely to be grounded successfully in other implementation domains, including enforcement. The rationale behind this is that our formalism is generic in the sense that it includes enough formal elements to be reduced to both ECA rules (monitoring with production systems) and state-based search (planning with PDDL), covering a wide range of expressiveness and complexity.

The second development of this chapter is a proposal for a SOA governance architecture that fulfills the industrial standards. An exhaustive analysis of the necessary components is included in this architecture, identifying data flows and repositories as well as computational elements that represent different roles in a governance system. This separation of concerns may prove useful not only as a guide of implementation and deployment but also as a starting point from which to make decisions regarding distribution and scalability, esp. in service-oriented architectures.

CHAPTER 9

Practical use cases

"Ma allora come possiamo fidarci della sapienza antica, di cui voi ricercate sempre la traccia, se essa ci è trasmessa da libri mendaci che la hanno interpretata con tanta licenza?"
"I libri non sono fatti per crederci, ma per essere sottoposti a indagine. Di fronte a un libro non dobbiamo chiederci cosa dica ma cosa vuole dire."[1]

Il nome della rosa
Umberto Eco

This chapter presents two of many applications in which our proposed language and monitor have been applied. The use cases chosen are: 1) the application of an organisational- and normative-centric approach for the design of distributed systems for eldercare management and 2) the integration of these computational elements to commercial games, more concretely as part of the reasoning of agents plugged into the games – the result of this integration was part of the use case demos shown at the end of the ALIVE Project (see Section 5.2).

The aim of the first use case is twofold: on one hand to show that the proposed norm language is expressive enough to model a complex, multi-party healthcare scenario; on the other hand to show how a norm based model provides a flexible and richer way to model and govern complex distributed processes. The aim of the second use case is to show the performance of the grounding of our operational framework grounded on an actual implementation under stress conditions.

[1]*'But then how can we trust ancient wisdom, whose traces you are always seeking, if it is handed down by lying books that have interpreted it with such license?'/'Books are not made to be believed, but to be subjected to inquiry. When we consider a book, we mustn't ask ourselves what it says but what it means.'*

9.1 Governance of situated agents in ambient intelligence

Eldercare management is progressively becoming a great concern due to the estimated growth in proportion of older population [Economic Social Affairs 2012], added to the fact that elder support is especially expensive [Economic Social Affairs 2004]. One of the most important common factors of eldercare management is that, for obvious reasons, most of it has to be done in a distributed fashion, away from healthcare institutions and typically at the patients' homes.

This distributed approach to daily care requires that elders be capable of autonomously taking several different medications at different time intervals over extended periods of time. This can easily lead to forgetfulness or confusion when following the prescribed treatment, specially when the patient is suffering multiple pathologies that require complex combinations of drugs. This gets worsened when elders suffer a cognitive impairment. Medication compliance is a critical component in the success of any medical treatment.

Assistive Technologies (AT) [1] have been recently providing successful solutions that help alleviate this problem. One of the main forms of application of AT consists in the interaction between several roles, such as patients, caretakers, relatives, health professionals, and last but not least, electronic devices – interfaces, sensors or actuators –, effectively taking the shape of distributed (agent) systems.

Such distributed nature has an organisational nature, as each role implies different objectives and available actions that affect the environment. Our use case is a contribution to the state of the art of this type of AT: the Coaalas project (COmpanion for Ambient Assisted Living on *ALIVE*-Share-*it* platforms) [Gómez-Sebastià, Garcia-Gasulla, and Alvarez-Napagao 2011], a framework for multi-agent systems that combines organisational and normative theories with Ambient Assisted Living (AAL) technologies. The project aims to create a society of organisational aware devices (typically sensors and actuators) that are able to adapt to a wide range of AAL situations.

Coaalas models the device network around the user as a society, including the set of behavioural patterns the devices are expected to follow. Coaalas effectively supports smart assistive tools that integrate human actors with the surrounding devices, contributing to the state-of-the-art in semi-autonomous and intelligent devices for elder people by allowing the devices to be both social- and norm-aware.

A summary of the architectural components surrounding Coaalas is depicted in Figure 9.1 [Gómez-Sebastià et al. 2015]:

- The *Social Network* is a layer that coordinates communication between users and between users and the rest of the software components, managing connections and relationships.
- The *Intelligent Layer* gathers and processed information from the rest of the modules and devices, generating relevant knowledge and enacting the services of the platform, i.e. monitoring, detection of patterns, generation of patient history,

[1] Assistive Technologies is an area of application of the state of the art in Artificial Intelligence focused on supporting and improving care activities in the context of elders or impeded patients.

etc. This layer, as reflected in Figure 9.1, is based on the ALIVE platform (see Figure 5.14) and will be the focus of the rest of this section.

- The *Middleware Layer* acts as a bridge between physical interfaces, as well as external systems such as healthcare information systems, and the Intelligent Layer.
- Physical *Interfaces* can be sensors and actuators such as a Smart Pill Dispenser, or user visual interfaces such as user applications, e.g. web browser or mobile app.

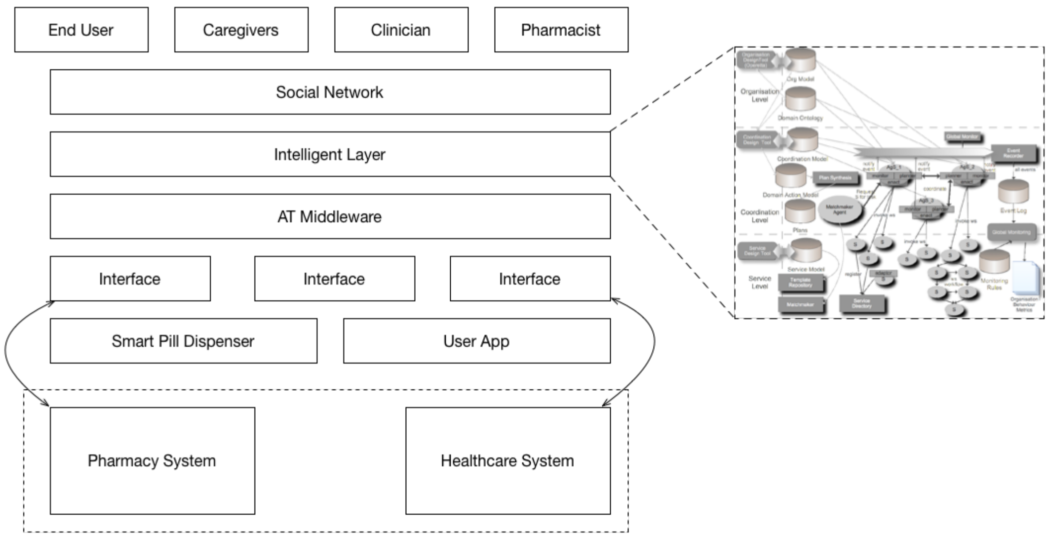


Figure 9.1: *Architecture of the system*

❧

9.1.1 Contribution: a social reminder for pills

In practice, Coaalas allows to integrate a wide range of sensors and actuators in a domotic setting, in order to transparently assist the user in their daily activities, while keeping all the participants of the healthcare workflow involved. Coaalas focuses on scenarios where the elder user, physically or cognitively impaired, has to comply with the medication prescribed by a doctor. Such scenarios can get especially complex due to a high and uncountable number of potentially probable circumstances, *e.g.*, the combination of several treatments that impose a temporal order on the doses, lack of user's discipline on taking the medicines during the correct interval, delays on the delivery of the medicines, lack of communication between the user and the doctor, and so on.

In such scenarios, the primary goal of our approach is to provide enough support to enable a change in the users' (including elders, doctors, health professionals among other stakeholders) non-compliant behaviors by engaging them in the drug intake task.

The first design and implementation of such a sensor/actuator in the context of Coaalas is the social electronic reminder for pills (see Figure 9.2) [Gómez-Sebastià et al. 2012], which tackles the supply of the required stock of medicines to users with difficulties to leave their house, while supervising that they follow the medical treatments prescribed by their doctors, not missing any dose due to forgetfulness or taking the medicines at the wrong time due to confusion. A summary of the related state-of-the-art on similar devices can be found at [Gómez-Sebastià et al. 2013].

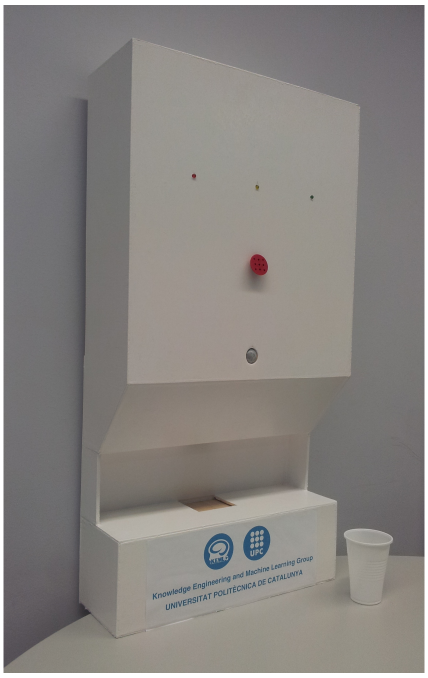

UPC
Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Figure 9.2: *Smart Pill Dispenser*

❧

This device supports the elderly or disabled people to manage their daily doses of medication while presenting the following three properties:

- *Social awareness*: The device is connected with other assistive devices and with relevant actors (such as doctors, caretakers and other health professionals, relatives, *etc*) for helping the elder take his daily doses of medication.
- *Autonomy*: The device can react to changes in the physical or social environment without requiring human intervention. Furthermore, it should be able to react to simple changes in the scenario autonomously (*e.g.*, a change in the scenario implies the pill dispenser is not filled by the patient any more, but by a care giver).

- *Normative awareness*: The device performs its task while following a set of specified behavioural patterns. However, due to its autonomy, the device has the option of breaking the patterns, provided it considers it will be in the benefit of the society (*e.g.*, if an incoming stock break is detected).

In particular, the three research questions addressed in this use case are:

1. Can a social-norm aware pill dispenser help elders adhere to their medication prescription? (i.e. daily take all the doses)
2. Can a social-norm aware pill dispenser help elders adhere to their medication regime? (i.e. daily take all the doses at the prescribed time and with the correct order)
3. Can a social-norm aware pill dispenser help the other users involved in the treatment workflow take care of unexpected events?

9.1.2 Modelling the system

Coaalas builds on the results of two European funded projects: EU-ALIVE (see Section 5.2) and EU-Share-*it* [Annichiarico and Cortés 2010] and provides a multi-agent platform able to integrate software agents embedded in the AAL devices and human actors. This allows for making AAL devices intelligent enough to organise, reorganise and interact with other actors. The agents embedded in the devices have an awareness of their social role in the system – their commitments and responsibilities – and are capable of taking over other roles if there are unexpected events or failures. Therefore, Coaalas creates a society of physically organisational-aware devices able to adapt to a wide range of AAL situations that could have an impact on the user's well-being.

As seen in Section 5.2, the ALIVE framework presents normative structures that allow for easily expressing both expected behavioural patterns and the actions to be taken when the actors involved in the scenario do not comply with these patterns. Substantive and constitutive norms allow the system to be flexible, by giving actors (human or computer-controlled) the choice to cause a violation if this decision is beneficial from an individual or collective perspective. For a full set of norms specifying the expected behaviour of the elements of the Coaalas system, including but not limited to the pill dispenser, see [Gómez-Sebastià 2016]. A set of selected norms is summarised in Section 9.1.3.

Apart from the normative (organisational) level, ALIVE also provides coordination structures (basically a repository of coordination plans automatically generated from the elements in the Organisational level) that provide actors' patterns of interaction, effectively allowing the system to move between relevant states (*e.g.*, the pill dispenser needs to be refilled, the pill dispenser has been refilled, etc.). The coordination structures are formed by tasks containing both pre and post conditions (*i.e.*, the state of the world before and after the task has been executed respectively) and the permissions required for executing the tasks (associated to the different roles in the scenario). A set of organisational-aware intelligent agents select a role according to their capabilities and start enacting the plans associated to that role as requested.

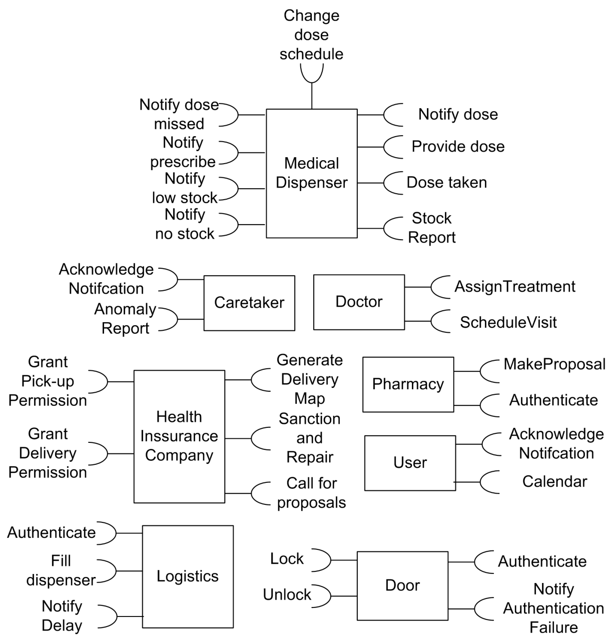


Figure 9.3: *Capabilities of the different actors*

❧

Finally, ALIVE also includes a service level that maps actions in the environment to abstract tasks. Non-organizational aware agents in the system register their capabilities (e.g., tasks they can perform) via a *white pages* system and are coordinated by the organizational aware agents to execute the tasks required for enacting the different plans. Figure 9.3 provides an example of actors' capabilities in a scenario that includes an intelligent pill dispenser.

9.1.3 Norm examples

COAALAS, at a conceptual level, defines a set of protocols that describe how the different agents of the system are expected to interact with each other. An example of such protocols is depicted in Figure 9.4, describing a full typical workflow from the patient's visit to the doctor to the dispensation of the doses.

Such a workflow, in the context of COAALAS, is implemented with regulative norms in the modelling phase and complemented in runtime with the use of constitutive norms, e.g. *John counts as a Doctor* or *bacon counts as a toxic substance* (see Figures 9.5 and 9.6). The set of norms defined by COAALAS is rather exhaustive, but in this section we show a subset of them, specifically selected to demonstrate the expressiveness of our

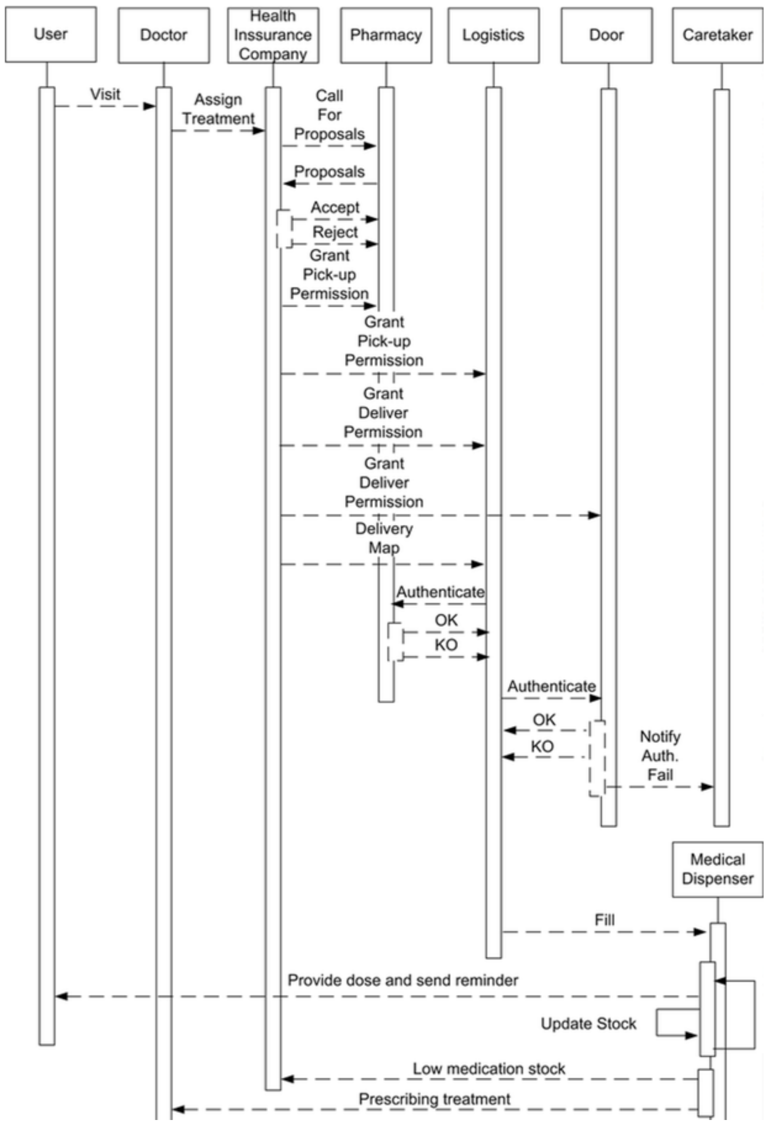


Figure 9.4: *Sequence diagram of a typical workflow, from prescription to dispensation*

❧

formalism presented in Chapter 5.

Norm 1 (see Figure 9.7) shows a typical achievement obligation, in which there are special[2] conditions for the activation, maintenance and expiration of the norm, as well as a repair action – checked against the repair condition – that has to be fulfilled if the norm is violated.

[2]Special in the sense that they are not trivial, i.e. they are not simply *true* or *false*.

Counts-as rule C_1: In the context of full cognitive capabilities of the patient A_i, with D_i being qualified to act as a doctor for at least a certain pathology p, if A_i has given D_i explicit authorisation for treating him/her for that pathology p, then D_i counts-as a *doctor of* A_i.

Context C_1	$requiresTreatment(A_i, p) \wedge isQualifiedDoctor(D_i, p) \wedge capable(A_i)$
Antecedent C_1	$authorises(A_i, D_i)$
Consequent C_1	$isDoctorOf(D_i, A_i)$

Figure 9.5: *Formal model for constitutive norm* C_1

❧

Counts-as rule C_2: In all instances, if a substance S_i has been declared as harmful by the Competent Authority, then S_i counts-as a *toxic substance*.

Context C_2	$\top$
Antecedent C_2	$officialStatement(S_i, consideredHarmful)$
Consequent C_2	$toxicSubstance(S_i)$

Figure 9.6: *Formal model for constitutive norm* C_2

❧

Norm N_1: Pharmacist $\phi_i \in \oplus$ is obliged to identify patient $A_j \in \mathcal{A}$ and take his prescription $R_k \in \mathcal{R}$ before delivering the medication $M_l \in \mathcal{M}$ to the patient. Otherwise, Competent Authority C sends a warning to pharmacist ϕ_i for violating the protocol specified.

Activation Condition N_1	$hasPrescription(R_k, A_j) \wedge isForMedication(R_k, M_l)$
Expiration Condition N_1	$hasDelivered(M_k, \phi_i, A_j)$
Maintenance Condition N_1	$\neg hasPrescription(\phi_i, R_k) \vee isIdentifiedBy(A_j, \phi_i)$
Repair Condition N_1	$warningSent(C, \phi_i)$
Timeout N_1	$\perp$

Figure 9.7: *Formal model for regulative norm* N_1

❧

In Norm 1, A_j is going to the pharmacy to pick up some medicines in order to refill his medical dispenser. e-Prescription systems are not available in the area where A_j lives right now, and some of the medicines are dangerous and therefore can only be dispensed with the corresponding medical prescription R_k. According to the protocols, the pharmacist ϕ_i has the obligation to retrieve the prescription and verify A_j's identity

Norm N_2: Patient $A_i \in \mathcal{A}$ has the obligation to follow the prescribed treatment $T_j \in \mathcal{T}$ since the date it starts until the date it finishes. $T_{j'}$ *counts-as* patient's treatment in the context of a patient A_i a treatment starting date $\tau_k \in$ T and treatment finishing date $\tau_l \in$ T. Otherwise, Competent Authority C sends a warning to patient ϕ_i for violating the protocol specified.

Activation Condition N_2	$isPrescribed(A_i, T_j, \tau_k, \tau_l) \wedge counts\_as(T_j, T_{j'}, A_i, \tau_k, \tau_l) \wedge actualTime(\tau_k)$
Expiration Condition N_2	$actualTime(\tau_l)$
Maintenance Condition N_2	$followsTreatment(A_i, T_j)$
Repair Condition N_2	$warningSent(C, A_i)$
Timeout N_2	$\perp$

Figure 9.8: *Formal model for regulative norm* N_2

❧

before delivering the medicines.

Norm 2 (see Figure 9.8) represents a maintenance condition: A_i has the obligation to follow the prescribed treatment for its duration. The smart pill dispenser allows to observe medical prescription adherences. The difference between an achievement obligation and a maintenance obligation in our framework is that in the former case the objective – the state they have to *see to it* that it is successfully achieved – of the agents is to reach the expiration condition without breaking the maintenance condition; while in the latter case the main goal is to fulfill the maintenance obligation until (if it happens, but it is not prioritary) the expiration condition holds. This difference, which might seem superficial *prima facie*, is in fact very important and carries important implications with it from a logic perspective [Governatori and Rotolo 2010]. For further discussion on the differences between dealing with maintenance obligations and dealing with achievement obligations in our framework, see Appendix A.

Norm N_3: Doctor $D_i \in \mathcal{D}$ is prohibited from accessing a medical record $\rho_j \in$ R if it belongs to a patient $A_k \in \mathcal{A}$ not assigned to him. Otherwise, Competent Authority C sends a warning to the doctor D_i for violating the protocol specified.

Activation Condition N_3	$\neg isDoctorOf(D_i, A_k)$
Expiration Condition N_3	$isMedicalRecordOf(\rho_j, A_k) \wedge isDoctorOf(D_i, A_k)$
Maintenance Condition N_3	$\neg AccessMedicalRecord(D_i, \rho_j)$
Repair Condition N_3	$warningSent(C, D_i)$
Timeout N_3	$\perp$

Figure 9.9: *Formal model for regulative norm* N_3

❧

Norm 3 (see Figure 9.9) is a special case of maintenance obligation that actually expresses a prohibition [3]. Doctors are prohibited from accessing the medical records of patients they do not have assigned. If the doctor accesses such medical records, he will be sanctioned with an official warning from the competent authority. Therefore prohibitions are modelled as negated states of the world to be maintained while the norm is active. Also, please notice that the norm activates (e.g. prohibition starts to hold) when the D_i is not assigned to A_k, and deactivates (e.g. prohibition does not hold anymore) with the assignment of D_i to A_k.

Norm N_4: Doctor $D_i \in \mathcal{D}$ assigns a new treatment $T_j \in \mathcal{T}$ to patient $A_k \in \mathcal{A}$ with a former treatment assigned $T_l \in \mathcal{T}$. The new treatment is applied for a time period between $\tau_m \in \mathrm{T}$ and $\tau_n \in \mathrm{T}$ whereas the old treatment comprises the period between $\tau_o \in \mathrm{T}$ and $\tau_p \in \mathrm{T}$. The constitutive norm that maps patient's treatment (T_q) is updated (dropping the old norm and adding a new one) effectively updating the commitments specified in *Norm* N_1.

Activation Condition N_4	$assignTreatment(D_i, A_k, T_j, \tau_m, \tau_n)$
Expiration Condition N_4	$hasTreatment(A_k, T_q, \tau_m, \tau_n) \wedge$ $\neg counts\_as(T_q, T_l, A_k, \tau_o, \tau_p) \wedge$ $counts\_as(T_q, T_j, A_k, \tau_m, \tau_n)$
Maintenance Condition N_4	$\top$
Repair Condition N_4	$\top$
Timeout N_4	$\perp$

Figure 9.10: *Formal model for regulative norm* N_4

❧

Norm 4 (see Figure 9.10) is a special case of achievement obligation with a trivial maintenance condition: the norm cannot be violated. In this case, the norm has been defined in order to automatically trigger a change in the normative state: A_k with diabetes visits D_i after a stomach operation. D_i prescribes a low-fiber diet, to be included in the actual A_k's prescription including medicines, exercise and a sugar free diet. Just like in S_1 patient has the obligation to follow the treatment with the same type of sanction to be applied if A_k does not abide. Notice how *constitutive norms* allow for updating the treatment (the institutional reality of the patient has changed) while the *regulative norm* N_1, without needing to be replaced or modified, still applies to the new scenario.

Norm 5 (see Figure 9.11) is a maintenance obligation actually representing a prohibition that can expire due to different reasons – hence the disjunction in the expiration condition – and also has a timeout set to fulfill the repair condition, which in this case represents a sanction directed to an individual (the patient) rather than an institutional

[3] As already seen in Chapter 6, our formalism does not explicitly include modalities. However, prohibitions and permissions can be expressed, with some limitations, by using the axioms of deontic logics. For further explanation, see Appendix A.

Norm N_5: Patient $A_i \in \mathcal{A}$ has the prohibition to consume substances $S_j \in \mathcal{S}$ that are institutionally considered toxic substances. Otherwise, the patient is required to contact their doctor immediately (before one hour later).

Activation Condition N_5	$isPatient(A_i) \wedge toxicSubstance(S_j) \wedge isDoctorOf(D_i, A_i)$
Expiration Condition N_5	$\neg isPatient(A_i) \vee \neg toxicSubstance(S_j)$
Maintenance Condition N_5	$\neg consumed(A_i, S_j, t)$
Repair Condition N_5	$contact(A_i, D_i)$
Timeout N_5	$actualTime(t + 3600)$

Figure 9.11: *Formal model for regulative norm* N_5

❧

repair action. A_i is elder and suffers from weak liver and is prohibited from consuming toxic substances, such as alcohol and tobacco. If the A_i takes them, considering the high risk of the behaviour, the event is logged in A_i's medical record, and will be presented to A_i's doctor during their next appointment. Please notice that the institutional definition of *toxic substance* can evolve through time via counts-as rule activations and deactivations, but thanks to the abstraction level provided by constitutive norms this prohibition will remain the same.

9.1.4 Adequacy of the norm language to the use case

In Section 2.2.3 we discussed about the limitations of the Service-Level Agreement languages proposed for the governance of service-oriented architectures due to their strong focus on defining constraints via simple rules or workflows. From a designer's point of view, such definitions (such as the one in Figure 9.4) can be either simple but rigid or exhaustive but too complex. For example, it is hard to define a maintenance obligation in a workflow when the expiration condition can potentially never happen.

The norm examples show a comprehensive set of different choices at the expresiveness level that our norm language allows that are not trivially transformable to rule- or workflow-based systems, or to distributed systems constrained by regimented norms:

- Maintenance and achievement obligations.
- Prohibitions as well as obligations.
- The possibility of defining an obligation as a trigger, effectively defining norm-driven regimentation.
- Institutional repair actions and agent-specific sanctions.
- Counts-as rules used to simplify the set of regulative norms.
- Timeout conditions allowing marking violations as failures.

Therefore, thanks to our approach, COAALAS supports expressing and introducing responsibilities at a high level with considerably less modification at the lower levels (e.g., reprogramming the agents in the different smart devices) because changes at higher levels automatically trigger changes at the lower levels. Additionally, using our language for norms allows for monitoring the different actions performed by the set of actors in order to fulfil the AAL (Ambient Assisted Living) tasks without having to define new regulations for each new device added to the platform. Deviations from the expected patterns of behaviour can be detected by directly querying the normative states, and repair actions can be read by a computer to enact enforcement. In summary, COAALAS provides support for dealing with unexpected events (e.g., sending a doctor, or an urgent shipment of medications to the patient when the pill dispenser device runs out of pills).

AT (Assistive Technologies) are applied to support people in their daily life. Most approaches focus solely on the direct interaction between users and the assistive tool. Approaches such as norm-based design have the potential to provide innovative mechanisms and methods capable of taking into account more complex interactions. For instance, the important role that third parties may have in user activities, and explicitly reflect the social constraints that apply in the relationship between device and patient. With COAALAS, smart devices are capable of reacting to deviations from the expected patterns of behaviour, effectively adapting to a wide range of AAL situations that could have an impact on the well-being of the user.

9.2 NORM-CONSTRAINED BEHAVIOUR IN FUN GAMES

Artificial Intelligence (AI) in commercial games provides the means to enhance the two-way communication with the human player by delivering the illusion of "intelligence" in the non-player characters' (NPCs) behavior [Millington and Funge 2009]. Although some specific types of AI algorithms, such as pathfinding or collision detection, have evolved to a mature state, the implementation of behavioral or strategical reasoning is, in most of the cases, still far from aligned with academic AI.

The current issues of commercial games AI are related to high-level concepts of gaming such as realistic virtual actors, automatic content and storyline generation, dynamic learning, or social behavior. Tackling these issues could represent a qualitative improvement on gaming experience from the player perspective and academic research on AI has good opportunities to provide solutions to these challenges[Charles 2003; Nareyek 2007].

Usually, AI in games encompasses a subset of academic AI techniques that implement *ad hoc* solutions in three groups[Millington and Funge 2009]:

i Movement mechanisms, providing the decision process to control NPC's motion, e.g. optimised real-time versions of A^* algorithms.

ii Behaviour control used to control NPCs' actions.

iii Strategy techniques used to co-ordinate groups of NPCs.

Whilst algorithms in (i) have evolved to mature state-of-the-art, solutions commonly used in commercial games for (ii) and (iii) are far from aligned with academic AI and are based on simplistic, rule-, automata- or case-based methods optimised for performance. These domain-dependent approaches present the following limitations in most of the cases:

- Blind specifications: the NPCs are programmed on *how* to act in reaction to environmental and/or other players conditions, but not *why* to act in a given manner; hence, the actions are purposefuless and, in most cases, not "natural" from the human player's perception.
- Lack of flexibility and adaptiveness: the rule-based actions are limited and reactive to external conditions, not beeing able to evolve, and providing reduced pro-activeness.
- Strange behaviour: the behaviour of the NPCs do no reflect the aspects of sociability and "participating in a whole", leading to unnatural actions from the human player's perception.
- Predictable behaviour: NPCs' tactics are easily discoverable by the human player and, after some time, predictable, leading to negative perception.
- Low reusability, as the solutions are commonly tailored to specific scenario domains and, therefore, not re-usable through different games even if they belong to the same genre.

Our hypothesis is that it is possible to create elaborate solutions for the issues of both individual behavior control and collective strategy techniques by integrating models based on organisational and institutional theories to control NPCs' behavior. This theory contributes to the systematic study of how actors behave within organisations. Hence, the actors in a game are described as an organisation which behavior is based on specific roles, norms, dependencies, and capabilities.

9.2.1 Related Work

There are already examples showing that higher levels of abstraction can be successfully used in commercial games' AI. Actually, some recent important commercial games such as *F.E.A.R*[Orkin 2006] or *Fallout 3*, have started to apply more complex cognitive patterns by using *GOAP* (Goal-Oriented Action Planning), a simplified and optimized version of *STRIPS* that allows for real-time planning of actions with pre- and post-conditions, even outperforming *Finite State Machine*-based algorithms in some scenarios[Long 2007]. Thus, these games execute complex symbolic reasoning not only about *how* to execute certain actions, but also about *what* to execute at each moment. We believe that, by using an even higher level of abstraction in order to reason also about *why* actions have to be performed, methods such as GOAP can be complemented and improved.

Adaptiveness in games has been already explored in academic AI research. However, existing approaches are either focused on individual reasoning[Leite and Soares 2006; Newell, Rosenbloom, and Laird 1987], or do not take into account high-level definitions that would allow for reasoning *why* to make a particular decision on a specific context[Spronck, Ponsen, and Sprinkhuizen-Kuyper 2006]. These approaches can get advantage of ALIVE by extending individual agents' reasoning cycle with *organisational awareness*.

In fact, organisational frameworks such as OperA[Dignum 2004] are already being explored for their use in *serious games*. In [Dignum 2008], organisational specifications are used to create a distributed intelligent task selection system that adapts to the player skill level and to model the storyline. With our work we intend to advance on this line of work by generalizing the use of organisational models for *fun games*, more focused on the realism of gaming experience, rather than on user modeling and learning.

The current issues of commercial games AI introduced in the introduction of this chapter are related to high-level concepts of gaming such as realistic virtual actors, automatic content and storyline generation, dynamic learning, or social behaviour. Tackling these issues could represent a qualitative improvement on gaming experience from the player perspective and academic research on AI has good opportunities to provide solutions to these challenges[Charles 2003; Nareyek 2007].

9.2.2 Proposal

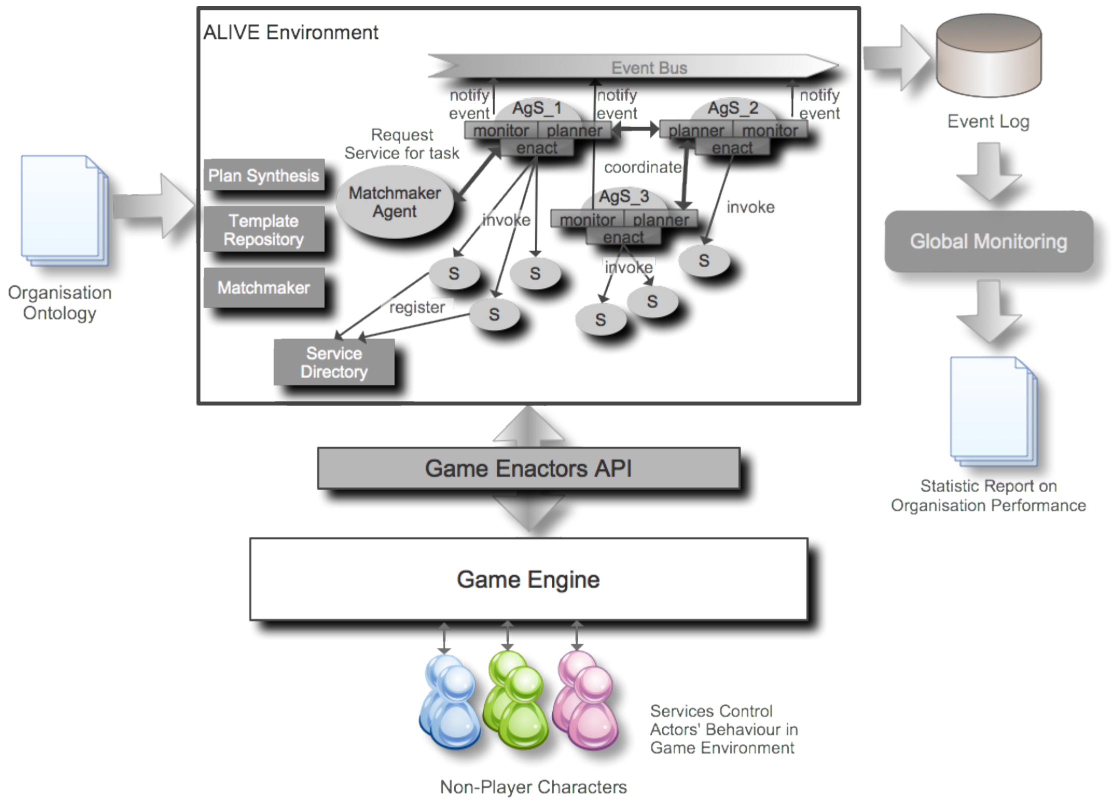


Figure 9.12: *ALIVE-Gaming coupling infrastructure*

❧

Our argument is to create elaborate solutions for the issues of (ii) behaviour control and (iii) strategy techniques by integrating the ALIVE framework to serious and fun games. This approach will provide extended flexibility to the elements that imply intelligent behaviour, e.g. actors and characters, teams of individuals, and narrative storylines. In addition, it will provide metrics that can be applied to evaluate the organisational behaviour using the games' environments as simulation scenarios. Hence, it would be possible to compare, learn, and improve NPC's behaviour with an approach based on organisation theoretical solutions for Game AI. This would contribute to overall flexibility and adaptiveness.

Figure 9.12 depicts the proposed architecture, which provides:

1. A practical solution to couple agents to the Game Engine, by defining the Game Enactor programming interface.
2. A tool to describe the Organisation Ontology, which contains a representation of agent structures.
3. The elements to describe game actors' behaviour via social structures based on norms, roles and their enactment, promoting the balance between autonomy and story direction.

We propose that this solution is applicable to both *fun games* and *serious games*. For the former, we foresee our solution helping to improve the games' actors' functioning with more flexibility and promoting natural behaviour. For the latter, the model reflects the socio-environmental behaviour of human societies, providing the basis for games and simulations that can be used in the emerging field of Computational Social Sciences. Next, we present a number of case studies in the applicability field.

9.2.3 Case Studies

(a) GTA IV (b) Warcraft III

Figure 9.13: *Games used as case studies*

We tested the solution in different games in order to validate our proposal, as depicted in Figure 9.13. Our intention was to analyse the advantage in terms of realism, flexibility and adaptability. Moreover, our application to simulation environments provides us with results useful for organisational research. In the case of commercial games, it requires access to the internal game control structures. We selected two representative examples for our case studies, considering the complexity and validity of achieved results:

- Warcraft III, from Blizzard.
- Grand Theft Auto IV, from Rockstar Games.

The working prototype of the *Warcraft III* connection was shown as one of the extra technology validation use cases at the end of the ALIVE Project.

9.2.3.1 *Real-time strategy games*

For many years, computer wargames have been designed as turn-based games. Real-time Strategy (RTS) games are an evolution of turn-based wargames, in which the player has to command a team of virtual individuals with diverse capabilities to achieve a common objective, commonly to defeat the teams of the human- or computer-controlled rivals. Other (sub)objectives include the capture and micro-management of resources, technological evolution, and so on. RTS games are interesting for our purpose in the sense that the concepts they deal with can be directly mapped to the ALIVE domain, i.e. organisational structure, roles, role hierarchy, objectives, and co-ordination.

From the AI development point of view, RTS games present two common issues:

1. Computer-controlled opponents become rapidly predictable and easily defeatable by using simple yet optimal strategies. NPC adaptation is rarely seen.
2. Although – at a high level of abstraction – the concepts and strategies of a RTS games are common to all of them, it is difficult to find AI solutions that can be reused, even between games from the same companies (see Figure 9.14).

```
function UpgradeEx takes nothing
 local unit u = GetTriggerUnit()
 local integer id = GetUnitTypeId(u)
 call DisableTrigger(trg_upgrade)
 call IssueImmediateOrderById(u, up)
endfunction
```

```
wait_build 2 forge
upgrade 1 p_ground_weapon 70
upgrade 1 p_plasma_shield 70
wait 2700
wait_build 1 cybernetics_core
upgrade 1 p_armor 70
upgrade 2 p_plasma_shield 70
wait 3600
```

```
rule getNextGathererUpgrade {
 int upID=kbGetCheapestUpgrade(tID);
 int pID=aiPlanCreate(id, planProg);
 aiPlanSetVariableInt(pID, upID);
 aiPlanSetDesiredPriority(pID, 25);
 aiPlanSetEscrowID(pID, cEscrowID);
 aiPlanSetActive(pID);
}
```

```
(defrule (goal 16 0)
  (can-research ri)
=>
  (release-escrow wood)
  (release-escrow food)
  (release-escrow gold)
  (release-escrow stone)
  (research ri))
```

Figure 9.14: *Example AI scripts for* unit upgrading*: (from left to right)* Warcraft III *and* Starcraft (Blizzard)*, and* Age of Mythology *and* Age of Kings (Microsoft)

❧

For these two issues, this is an scenario that could very well benefit from the adaptability offered by the ALIVE infrastructure and serve as a useful proof-of-concept. RTS games are also interesting for our purpose in the sense that the concepts they deal with can be directly mapped to the ALIVE domain, i.e., organisational structure, roles, role hierarchy, objectives, and coordination. We aim to produce computer opponents capable of adapting to unpredictable scenarios by dynamically improving at the organisation and coordination layers. Moreover, this type of game would provide us a clear visual interface to execute simulations of organisations in real-time.

Modeling. We modeled the organisational specification of an abstract RTS game (see Figure 9.15 and Figure 9.16). This specification identifies the set of stakeholders, the goals of each of them, landmarks and scenes related to those goals, and the normative structure of the system. The set of stakeholders includes *commander*, *medic*, *explorer*, *worker*, *attacker*, or *defender*. These are directly mapped to *roles*.

The normative structure included the following norms:

- Soldier units count as defenders.
- Soldier units count as attackers.
- Peasant units count as resource gatherers.
- In the case of the presence of enemies in the base, peasants count as defenders.
- It is forbidden to create military units until there are enough workers to support them (see Figure 9.17).
- All defenders are obliged to attack enemies that are in the base.
- All atackers are obliged to attack any enemy seen in the map.
- Peasants are obliged to gather resources whenever they are idle.
- The amount of a type of resource stashed cannot be larger than twice the amount of another type of resource.

One of the main benefits of applying institutional structures to this type of game is that these high-level specifications can be reused through different RTS implementations, solving the issue of different types of NPCs enacting very similar roles across different games.

For each *role*, its *goals* were identified, as well as the hierarchical relationships among the set of *roles*. Figure 9.15 shows the roles (nodes) and the hierarchical relations between them (edges) in a graph-like representation. For instance, in order to fulfill the objective *Produce_new_worker*, the objective *Gather_Gold* has to be taken care of. As *Produce_new_worker* is pursued by the role *Unit_producer*, and *Gather_Gold* by the role *Gold_Gatherer*, a relationship between the roles *Unit_producer* and *Gold_Gatherer* is created, because the first role is dependent on the second one for the fulfillment of its objective. For each objective, a *state description* is modeled, representing the state of the world where the objective has been fulfilled.

To define how each of these goals must be accomplished, *landmarks* are defined: for each objective, a set of ordered landmarks which must hold true in order to achieve

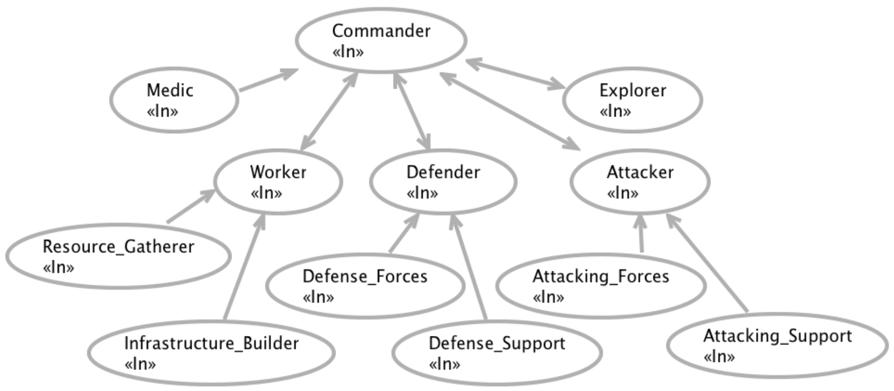


Figure 9.15: *Social structure for generic RTS games (*OperettA *Tool screenshot)*

❧

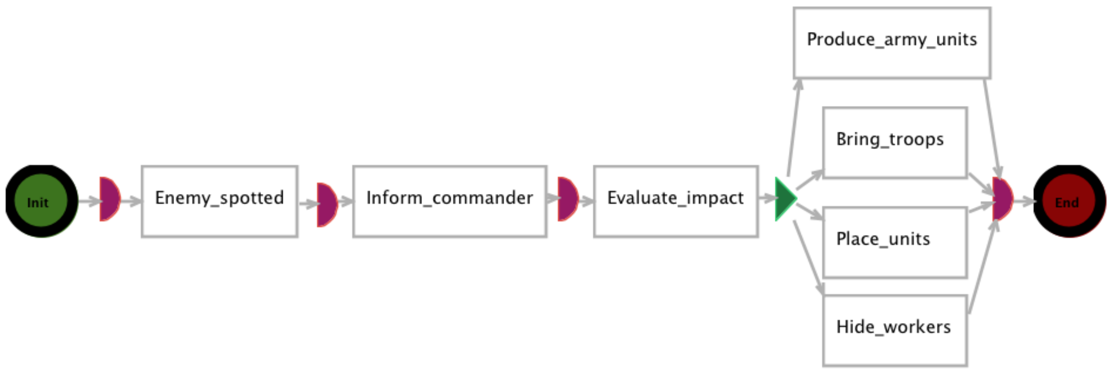
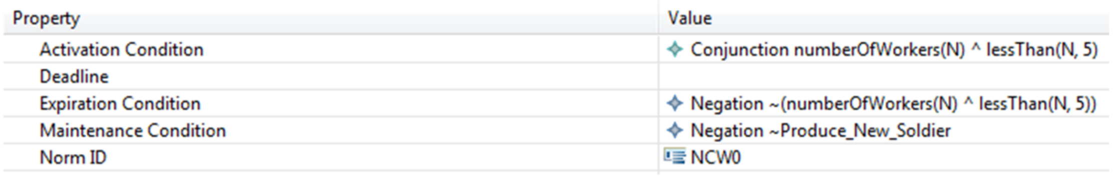


Figure 9.16: *Interaction structure for* Defend city *(*OperettA *Tool screenshot)*

❧

a certain goal defines a *scene*. For each scene, an instance of its execution in the actual environment entails a certain state of one or more objectives. *Figure* 9.16 shows an example of scenes (nodes) with the transitions between them (edges) in a graph-like representation. For instance, when the objective *Gather_wood* is fulfilled, the landmark *Wood_Gathered* is reached. Role *Wood_Gatherer* is involved as the landmark *player* because it has the objective *Gather_wood* assigned.

The last element to be defined on the organisation level is the set of norms. Norms are defined by the *activation*, *maintenance* and *expiration* conditions defined in Chapter 5, modeled as *partial state descriptions*. Figure 9.17 shows an example of a norm modeled for the use case: *it is forbidden to produce a soldier unless 5 workers are already available*.



Property	Value
Activation Condition	✦ Conjunction numberOfWorkers(N) ^ lessThan(N, 5)
Deadline	
Expiration Condition	✦ Negation ~(numberOfWorkers(N) ^ lessThan(N, 5))
Maintenance Condition	✦ Negation ~Produce_New_Soldier
Norm ID	NCW0

Figure 9.17: *Norm example applied to our case* (OperettA *Tool screenshot)*

❧

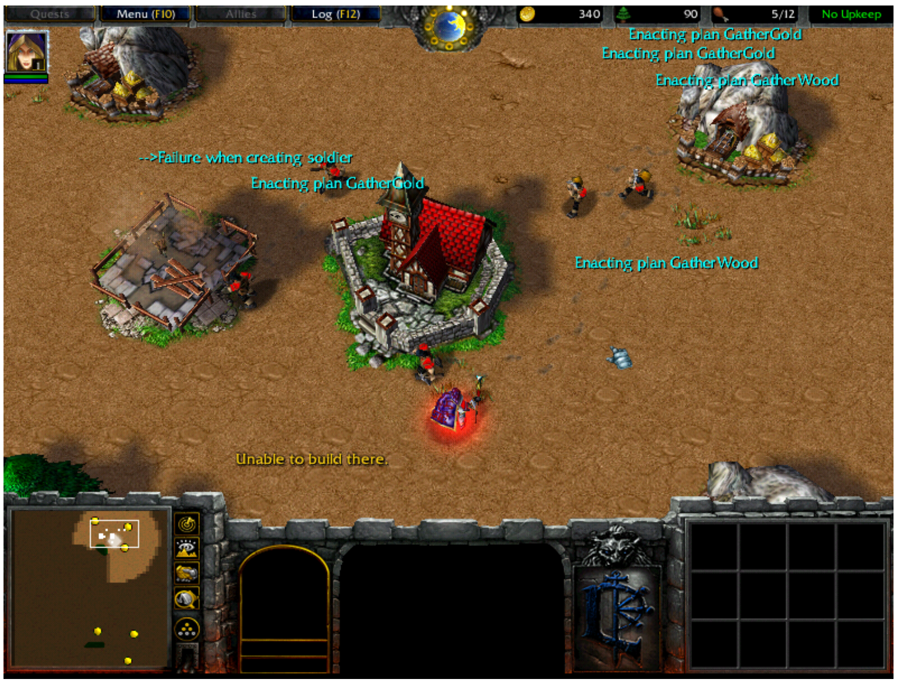
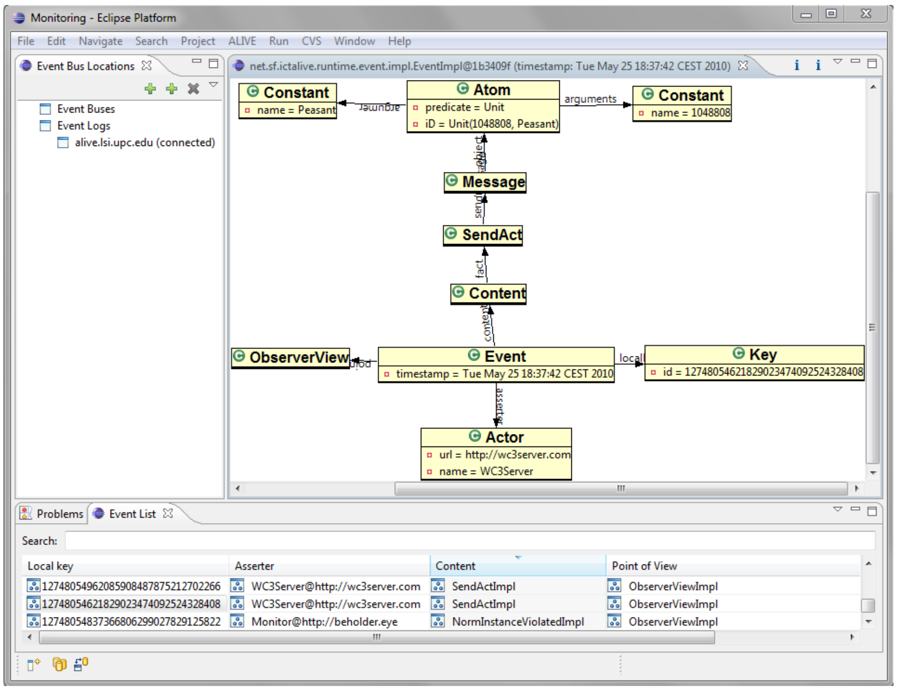


Figure 9.18: *Warcraft III units enacting actions sent from the ALIVE platform (game view)*

❧

Implementation. We designed an intelligent agent that is connected to the *Warcraft III* game through a Game Enactor, allowing for bidirectional communication via sockets. The agent is organisational-aware by reading and integrating the ALIVE organisational and coordination models into its reasoning cycle.

The low-level events are obtained through the Game Enactor, and our normative monitor environment provides mechanisms for the interpretation of these event, providing organisational meaning. Thus, the agent is capable of perceiving the "state of the world", reacting to events happening in the game at runtime, e.g. a unit being created,


Monitoring - Eclipse Platform
File Edit Navigate Search Project ALIVE Run CVS Window Help
Event Bus Locations
Event Buses
Event Logs
alive.lsi.upc.edu (connected)
net.sf.ictalive.runtime.event.impl.EventImpl@1b3409f (timestamp: Tue May 25 18:37:42 CEST 2010)
Constant
name = Peasant
argument
Atom
predicate = Unit
iD = Unit(1048808, Peasant)
arguments
Constant
name = 1048808
object
Message
sender
SendAct
fact
Content
content
ObserverView
pov
Event
timestamp = Tue May 25 18:37:42 CEST 2010
local
Key
id = 12748054621829023474092524328408
asserter
Actor
url = http://wc3server.com
name = WC3Server
Problems
Event List
Search:

Local key	Asserter	Content	Point of View
12748054962085908487875212702266	WC3Server@http://wc3server.com	SendActImpl	ObserverViewImpl
12748054621829023474092524328408	WC3Server@http://wc3server.com	SendActImpl	ObserverViewImpl
12748054837366806299027829125822	Monitor@http://beholder.eye	NormInstanceViolatedImpl	ObserverViewImpl

Figure 9.19: *Warcraft III units enacting actions sent from the ALIVE platform (monitor view)*

❧

or a soldier spotting an enemy, and of reasoning about which actions should be taken in the game, taking into account the current state of the world and the organisational constraints (e.g. objectives and normative constraints). Also, an agent may (or may not, depending on the individual utility) decide to discard a particular action if a norm is forbidding to enact it given the current state of the world.

The Game Enactor allows agents to enact actions in the game (see Figure 9.18 and Figure 9.19). Once the reasoning process has decided which are the next actions to be performed, agents are able to communicate with the game, making the unit responsible of each action to enact it according to the role and plan structures defined in the organisational specification.

Currently, agents are implemented in Java on top of the *AgentScape* multi-agent platform. These agents are organisational aware, and are capable of planning and enacting plans based on ALIVE models. The planning process can be adjusted by an internal configuration module called Plan Rules.

This Java service is used by the ALIVE framework. With this implementation, agents are able to:

- Perceive the "state of the world", reacting to events happening in the game at runtime, e.g. a unit being created, or a soldier spotting an enemy.
- Reason about which actions should be taken in the game taking into account the

current state of the world.

- Include the ALIVE specification in their reasoning. Agents will take into account the organisational structure: roles, plans and norms, defined in the ALIVE model, and proactively decide at each moment which actions to enact in order to accomplish the organisational objectives. The agents may also decide to discard some actions to be enacted if an organisational norm is forbidding to enact them given the current state of the world.
- Enact actions in the game (see Figure 9.18). Once the reasoning process has decided which are the next actions to be performed, agents are able to communicate with the game, making the unit responsible of each action to enact it according to the role and plan structures defined in the organisational specification.

This is the scenario that could best benefit from the adaptability offered by the ALIVE infrastructure. A common issue of RTS games is that after some amount of time spent on it, computer opponents are predictable and easily defeatable by using simple yet optimal strategies. We aim to produce computer opponents capable of adapting to unpredictable scenarios by dynamically improving at the organisation and coordination layers. Moreover, this type of game would provide us a clear visual interface to execute simulations of organisations in real-time.

9.2.3.2 *Sandbox game: GTA IV*

First we will test our environment on sandbox games, also known as free-roaming games. In these kind of games, players are given a large amount of freedom, with non-linear storylines and different paths to completion. For example, the Grand Theft Auto (GTA) series allows the player to wander around a whole city and interact with hundreds of NPCs and objects.

In *free roaming* games such as GTA, most of the interactions with characters are scripted, giving the player a feeling of repetitiveness after a few hours of play. On the other hand, the higher-than-normal freedom given to the player also provides less realism.

Our objective was to define a high level social structure, simulated by the ALIVE coordination layer, with dynamic adaptation of interaction patterns, using GTA as the graphic interface of such a social environment. For example, in GTA the player is almost free to behave in a violent way while driving a car. Passing red lights, driving in the wrong direction, and running over people are actions that have no consequences in the vast majority of cases. We implemented a prototype by designing, at the organisational level, traffic norms and roles defining authority figures, i.e. police (see Figure 9.21). Police agents plan and reason about the sanctions to apply when detecting a traffic norm violation, which can consist on imposing a fine or initiating a car chase, depending on the gravity of such violation.

For the GTA connection with ALIVE we used *GTA ScriptHook*, an open-source tool which allowed us to capture all possible events and execute all possible actions in the

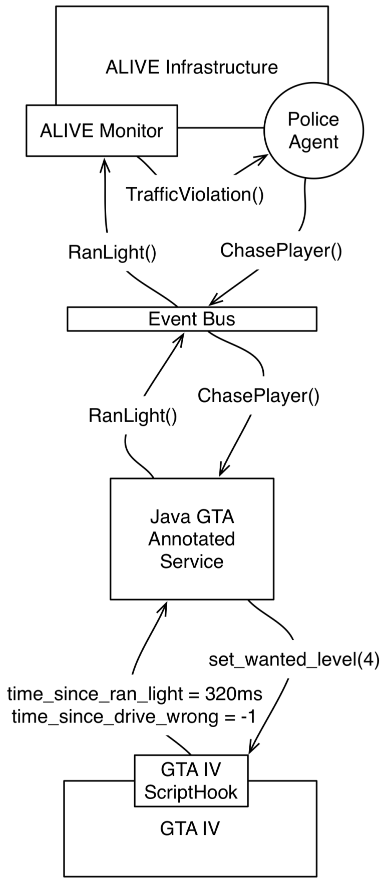


Figure 9.20: *ALIVE-Gaming coupling infrastructure for GTA*

❧

game running environment, including the control of NPCs' behaviour. The following is a typical flow of information on our system (see Figure 9.20):

1. An event, i.e. running past a red traffic light, happens on the game.
2. *GTA ScriptHook* captures the event and provides it to the *Java GTA Annotated Service*.
3. The service interprets the game event as a low-level event and puts it on the *Event Bus*.
4. The *ALIVE Monitor* captures the event from the *Event Bus* and infers its high-level interpretation.
5. This interpretation triggers the generation of a new event *ChasePlayer* that is registered in the *Event Bus*.
6. The *Java GTA Annotated Service* captures the *ChasePlayer* event from the *Event Bus* and, via *ScriptHook*, modifies the game.

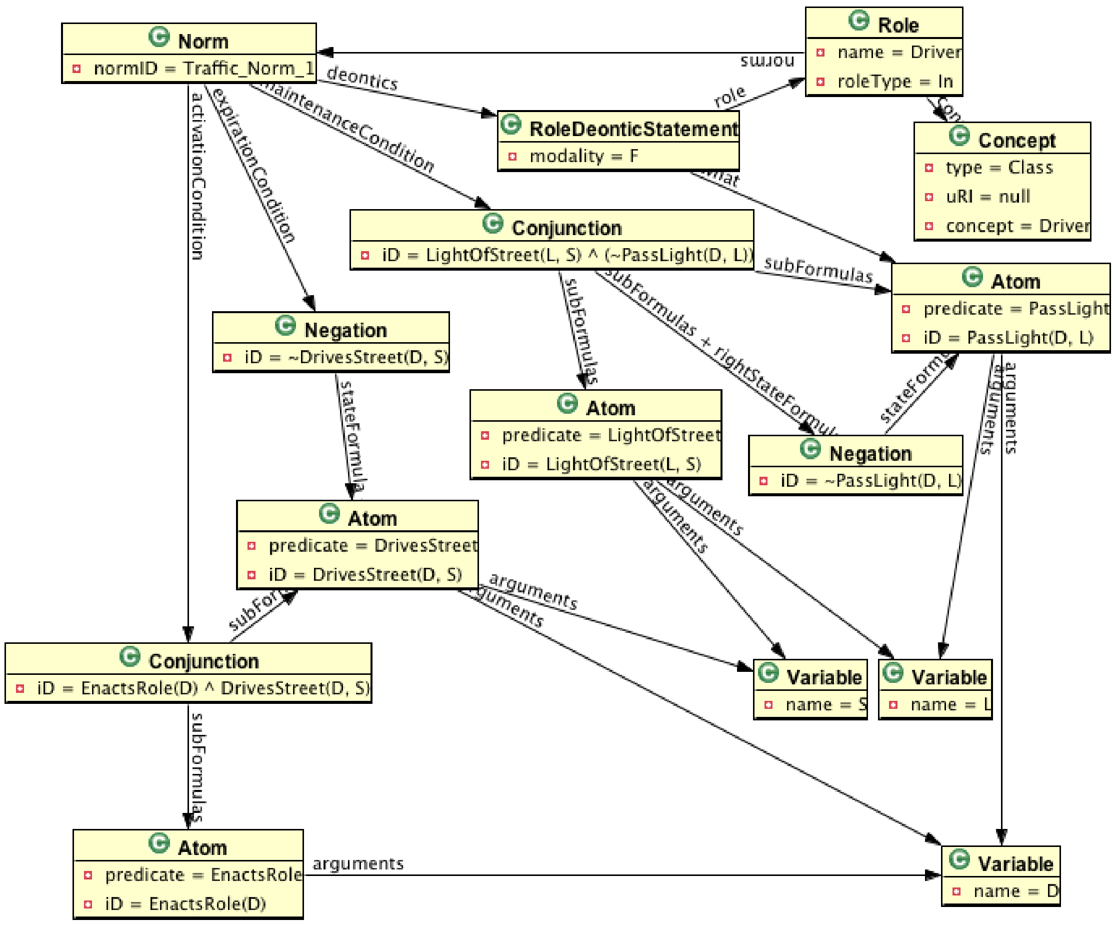


Figure 9.21: *Graphical representation of the norm* it is forbidden to pass under a red light (OperettA *Tool)*

❧

7. This will effectively make something happen on the game, i.e. player being chased by police forces, as a response of having run a traffic light.

As we have already seen in Section 5.2, norms modelled using the ALIVE tools are not regimented but substantive, which means that the player –as well as any NPC– can decide not to fulfill them. Thus, a player can decide to break traffic rules if the police is not around or at line of sight, or if the player has no concerns about the possible sanctions enforced by the NPCs. This is a simple example, but more complex examples can be designed to create obstacles or motivations for the player, by reasoning at runtime about the social-environmental context in the game at a certain point of time.

Our intention was to design a full set of organisational constraints, i.e. norms, individual objectives and roles, in order to define high-level social structures in the game, therefore improving realism through sensible and adaptive interactions with NPCs.

The working prototype of this scenario consisted on a type of agent that represented all policeman generated by the game, and which received inputs from the system based on limited observability (the behaviour of the player was not tracked if the player was too far and/or not in the line of sight). There were no counts-as rules in this case, and norms dedicated to traffic rules not being observed by the original game engine:

- It is forbidden to pass through a red light.
- It is forbidden to surpass 110 miles per hour.
- It is forbidden to crash against a car.

This scenario, in terms of visible benefits for demonstration, was more limited than the one based on RTS. However, with proper constitutive rules and a more varied set of regulative rules, this type of scenario is potentially very interesting for organisational/institutional simulation purposes. Currently, we have abandoned *GTA IV* as engine and we are adapting this scenario to the *World of Warcraft* game, as it provides extra challenges to tackle such as having huge amounts of events per second to be processed by the normative monitor.

9.2.4 Experimental results

In order to check the validity of our normative monitor from an efficiency point of view, we analysed traces of data coming from an open-source server of *World of Warcraft*[4]. Our modification of the game allows us to use norms from a Real-Time Strategy Game domain into a *sandbox-like* multiplayer game such as *World of Warcraft*. This modification was made with the objective of exploring emergent narrative with social/institutional components (for more information, see [Alvarez-Napagao et al. 2012]).

```
(eval '(defrule norm-instance-fulfillment
         "norm␣instance␣fulfillment"
         [wire.preds.Expiration (= ?n norm) (= ?f formula)]
         [?ni <- wire.preds.Instantiated (= ?n norm) (= ?theta substitution)]
         [wire.preds.SubsetEQ (= ?theta2 subset) (= ?theta superset)]
         [wire.preds.Holds (= ?f formula) (= ?theta2 substitution)]
         =>
         (do
           (retract! ?ni)
           (insert-unconditional! (->Fulfilled ?n ?theta)))))
```

Figure 9.22: *Example of translation of base rule to* clara

❧

For these experimental results, we use an advanced version of the *Game Enactor* that uses *clara*[5] as a rule engine. *clara* is a rule engine very similar to Drools in syntax and

[4]The implementation of this server is TrinityCore and our especially tailored version can be found at https://github.com/kemlg/trinitycore-conciens

[5]https://github.com/rbrush/clara-rules.

design concept and is written from scratch in Clojure. The success in grounding our formalisation to *clara* can also be seen as additional empiric proof of the flexibility given by using production systems as the base implementation semantics. Figure 9.22 shows an example of base rule written in *clara* and can be directly compared to the rule "*norm instance fulfillment*" listed in Figure 7.9.

Figure 9.23 contains an example Clojure structure representing some example norms mentioned in Section 9.2.3.1. This structure is created automatically from OperA models and is transformed into rules following the procedure described in Section 7.3.2.

Our modified *World of Warcraft* server captures every internal event related to a change in the physical environment of the game and sends them in JSON format to the *Game Enactor*, which fires the *clara* rule engine instance after every event, in order. For experimentation purposes, we inject 450 autonomous bots that act as regular players and carrying out random actions, such as buying, gathering resources, or killing.

Figure 9.24 shows the relationship between number of events generated and time taken by the game to generate them, out of 522 samples parametrised by random event amounts. As can be seen from the chart, if outliers are ignored, the speed of the generation of events is fairly consistent. From the linear regression obtained from this data we can infer that the speed of generation is of approximately 14.4179436900546 events per millisecond (14417.94 events per second). This means that in an average case the ideal processing speed of our rule engine for this game should be around 69.36 microseconds/event.

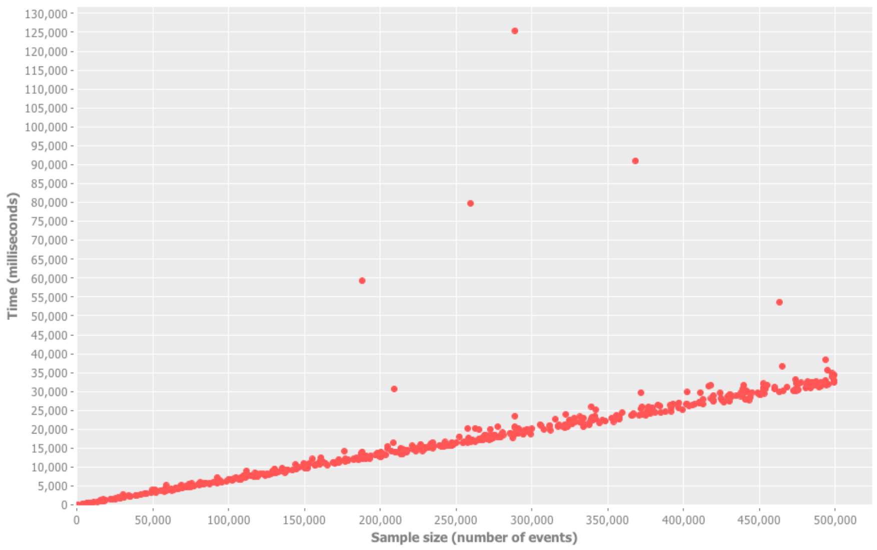


Figure 9.24: *Scatter plot of events samplings and time spent per sample*

❧

While this speed is not particularly high – systems such as Kafka or RabbitMQ are

```
{:norms [{:norm-id "nCW0",
          :conditions
          {:expiration {:type "disjunction",
            :formulae
            [{:type "conjunction",
              :formulae
              [{:type "negation",
                :formula {:type "predicate",
                          :name "NumberOfWorkers",
                          :arguments [{:type "variable", :name "x78"}]}}]}
             {:type "conjunction",
              :formulae
              [{:type "negation",
                :formula
                {:type "predicate",
                 :name "lessThan",
                 :arguments [{:type "variable", :name "x78"}
                             {:type "constant", :value "5"}]}}]}]},
           :activation
           {:type "disjunction",
            :formulae
            [{:type "conjunction",
              :formulae
              [{:type "predicate",
                :name "NumberOfWorkers",
                :arguments [{:type "variable", :name "x78"}]}
               {:type "predicate",
                :name "lessThan",
                :arguments [{:type "variable", :name "x78"}
                            {:type "constant", :value "5"}]}]}]},
           :maintenance
           {:type "disjunction",
            :formulae
            [{:type "conjunction",
              :formulae [{:type "negation",
                          :formula
                          {:type "predicate",
                           :name "Produce_New_Soldier",
                           :arguments []}}]}]}}}],
 :cas-rules [{:context "Universal",
              :concrete-fact
              {:type "disjunction",
               :formulae [{:type "conjunction",
                           :formulae
                           [{:type "predicate",
                             :name "Unit",
                             :arguments
                             [{:type "variable", :name "x85"}
                              {:type "constant", :value "Peasant"}]}]}]},
              :abstract-fact
              {:type "disjunction",
               :formulae [{:type "conjunction",
                           :formulae
                           [{:type "predicate",
                             :name "Worker",
                             :arguments [{:type "variable", :name "x85"}]}]}]}}]}
```

Figure 9.23: *Example norms in Clojure*

❧

	1 norm	10 norms	100 norms	1000 norms
Simple formulae	1.859195	5.653881	79.379386	177.226990
Complex formulae	2.526412	15.937769	98.955428	193.541208
Deficit (events/microsecond)	–	–	10.02	107.87

Table 9.1: *Results of the experiment (in microseconds)*

❧

designed to cope with one order of magnitude higher – it is still decent for testing a system with a complex business logic such as the monitoring of norms – as opposed to more simple business logics such as message brokering or log storage. The design of our experiment is:

- Several amounts of norms: 1 / 10 / 100 / 1000.
- Different norm complexity: no disjunction in the formulas / each formula contains 3 clauses.

We run a benchmark for each cartesian product of both parameters, each benchmark consisting in bootstraping the rule engine and executing 60 tests of 50000 events each, while connected to the live output stream of the game. The results of this testing set are summarised in Table 9.1, each cell of the first two rows containing the average execution time per cycle – insert event into the rule engine + fire the rule engine + store the normative state – in each scenario. The third row contains the trailing number of events per microseconds that our rule engine is not able to pick up.

One of the most immediate conclusions we can take from our experiment is that if we add norms – regardless of the complexity of their formulas – we will eventually reach a point where the rule engine is slower than the data gathering speed. This is similar to the results of other more complete benchmarks such as [Xiao and Zhong 2010]. This is due to the complexity of the RETE algorithm [Albert and Fages 1988]: $\mathcal{O}(log(P))$ in the best case and $\mathcal{O}(P \cdot W)$ in the average case, where P is the number of rules and W is the number of facts. This confirms that rules are the dominant input parameter when measuring complexity – input (*brute*) facts are only interesting for us in order to generate institutional facts, so we can discard them if necessary.

Therefore, there is an opportunity to find ways to reduce the inherent complexity when having larger sets of norms. This is already being explored by applying horizontal distribution of the norm condition clauses, and [Gómez-Sebastià 2016; Gómez-Sebastià, Alvarez-Napagao, and Vázquez-Salceda 2011] are a direct result of this line of work.

On the other hand, the conditions of the experiment are close to a stress test. In most commercial games, events that can be retrieved via scripting, software development kits or APIs are heavily pre-filtered and it is extremely rare to find a case in which we need to process so many events per second. Actually, in the worst case scenario from the ones summarised in Table 9.1 (1000 complex norms), our monitor can process more than 5000 events per second, which is more than enough for most cases.

9.2.5 Adequacy of our operational formalisation to the use case

We suggest that the combination of our norm language and its grounded operationalisation contribute to Game AI solutions by providing an adaptive, extensible, flexible and efficient solution for norm monitoring to the game development industry. The main advantage of this approach is that, through norm-based design, developers can specify NPCs' behavior in terms of *why* they should do something, not only *what* and *how* to do it: actors in a game are described as an organisation whose behavior is based on specific roles, norms, dependencies, and capabilities. Our solution provides a methodology and tools for developers to model gaming scenarios using social structures.

The use of high-level norms specifically contributes to the Game AI issues of behavior control and strategy techniques, by providing:

- open specifications where NPCs are programmed in terms of why they must act in a certain way;
- enhanced flexibility and adaptiveness by describing NPC's behavior based on organisational terms;
- more *natural behavior* as NPCs may act autonomously, respecting environmental conditions and organisational objectives that will be perceived as *natural*; and
- improved reusability, as the proposed solution is generic and can be attached to a variety of commercial games through a common interface and customised organisational models.

9.3 Conclusions

In this chapter we have seen two use cases that help illustrate how our contributions presented in Part II can be applied in real-world scenarios.

First of all, we have explored Coaalas, a distributed Assistive Technology platform that allows specifying the relationship between doctors, patients, caregivers and so on as well as the constraints that affect such relationships in a flexible and adaptive way. In this scenario we have focused on demonstrating the expressiveness of our norm language.

Second, we have presented a monitor for fun and serious games with the objective of being able to model organisational and social relationships between non-player controlled characters, and between these and human players. We have used this use case to discuss the efficiency of the grounding of our operational formalisation.

CHAPTER 10

Conclusions

Ces paysages d'eau et de reflets sont devenus une obsession. C'est au-delà de mes forces de vieillard, et je veux cependant arriver à rendre ce que je ressens. J'en ai détruit... J'en recommence... et j'espère que de temps de choses il restera quelque chose.[1]

CLAUDE MONET

This dissertation has tackled an issue that will probably become of great importance in the next few years due to the consolidation of distributed systems as the foundation of *computation of interaction*. As the number of such systems and their complexity grow, there is a challenge in trying to coordinate them in a way that does not compromise their internal implementation in the form of hardcoded constraints. Agents and services, in this paradigm, are to be seen as members of socio-technical systems with mixed human and artificial entities interacting together, or even as members of institutions that can emulate the human ones with the use of high-level behavioural constraints.

In Part I, we have answered Research Question 1 – *What properties can we infer from individuals in distributed systems that allow us to treat them as part of a socio-technical system?* – by exploring in more detail the concepts related with this challenge.

First of all, we analysed *computation as interaction* in Chapter 1, identifying the individual elements of distributed systems, intelligent agents and services, as agents: capable of agency and therefore susceptible of being coordinated using mechanisms analogous to those already existing in human organisations.

In Chapter 2, we explore one of such mechanisms: normativity. We stress the importance of having both constitutive and regulative norms, because that allows us to have proper institutions as our social abstraction for distributed systems. Also, in this

[1]*These landscapes of water and reflections have become my obsession. They are far beyond my old man power and, in spite of everything, I want to succeed in rendering what I feel. I have destroyed...I start over again...And I hope something will eventually remain.*

chapter we summarise several ways to implement norms in a computable way: Deontic Logic, Event-Condition-Action (ECA) rules and Service-Level Agreements. They have different levels of complexity and directly proportional expressivity capabilities, and therefore we conclude that the perfect scenario would be to be able to reduce a language with high expressivity to a formalism with the least possible computational complexity.

We then explore in Chapter 3 the systems that use the abstraction of norm in two manners: as institutions, grounded in theoretical research, and as SOA governance, created with pragmatism as rationale.

Summarising our study of the state of the art, we detect the need to have a language allowing the highest possible expressivity, a combination of deontic logics and counts-as rules, and a proper reduction to production systems, which are based on ECA rules. This is the focus, as well as the result, of our work presented in Part II and Part III.

10.1 Contributions

Our Research Question 2 – What mechanisms can we provide to distributed systems so that they can properly act as socio-technical systems? – is tackled in Part II. Taking into account the related work already present in the literature, our answer is based on filling the gaps described in Chapter 4 by taking a pragmatic approach based on real-world applications' requirements.

Our first contribution in this direction, presented in Chapter 5, is **a language of norms with support for constitutive rules, tailored for distributed systems**. This language has roots in several theories, such as deontic logics and speech acts, while still being translatable to computer-based languages (XML, XMI, Lisp). Along with the language, we presented several layers of communication and interaction protocols that can be used – but not mandatorily – by agents in distributed systems. This language was created for its use especially in service-oriented architectures. In future work, this language could be improved by adding some missing deontic-related terms to the vocabulary and thus to the operationalisation, such as *power*.

In Chapter 6 we present our second contribution: **operational semantics covering both constitutive and regulative norms, with a grounding on deontic logics**. An important contribution with respect to existing proposals is that norm instances are not only being tracked – they are not even tracked in many proposals – but they become first-class objects in our model. This allows agents to reason not only about the norm itself but also about the variables that triggered a particular norm instance. We also present two separate lifecycles for norms and for norm instances, a distinction that is usually confused in the literature when they are concepts that reside in different levels of abstraction.

An important thing to note is that we provide, in Appendix A, **reductions from our operational semantics to several flavours of deontic logic**. This is especially relevant if we take into account that we are augmenting our formalism to be able to cope with norm instances, which are not a part of Standard Deontic Logics.

However, our operational semantics can still be largely improved. On one hand, the objects of the norm conditions are right now formalised in propositional logic. Other options, such as first-order logic – which are available in many rule engines – or LTL should be explored. Also, our norm instance lifecycle, which is adequate for most uses, is rather limiting due to having its roots in LTL. Other logic frameworks should be tested in this respect, such as CTL or CTL*.

Finally, in Chapter 7 we present our **normative monitor, based on a reduction of our operational semantics to production systems**. Having all our operational semantics reduced, at the implementation level, to simple rules executable in any rule engine provides our normative monitor with the capability of being used in virtually any computation environment. This tackles directly our objectives of 1) enabling the use of norms in a wide range of distributed system types, including services (and microservices); and 2) effectively decoupling agent reasoning from the detection of normative states and thus providing social reality in real-time scenarios.

Another relevant novel outcome, shown in Section 8.1, is that our operational semantics are not only reducible to production systems but also to planning languages. This effectively means that an agent planning its behaviour based on a normative context and executing it successfully will produce a social reality that will be exactly the same as the social reality detected by an agent using the normative monitor.

As with the rest of the contributions, there are important upgrades that can be made to our normative monitor. The most obvious one is giving it semantics for dynamic change of normative context. Because the underlying technology used is production systems, this is entirely possible and is already being explored in [Gómez-Sebastià and Alvarez-Napagao 2012].

Additionally, in Part III we have explored ways to make our contributions usable not only from a monitoring but also from an enforcement perspective, and we have presented practical examples that show the applicability of our contribution in real-world scenarios. In fact, we have presented **an architecture for norm enforcement based on our operationalisation and implementation to fulfill the requirements of a SOA governance system**.

Overall, our contributions represent a self-contained package of computational elements for the monitoring of social state that can be used in distributed systems without much effort and with a reduced time complexity. Therefore, the field of socio-technical systems can benefit from the language and the normative monitor by enabling the incorporation, to such systems, of software components with very low implementation complexity, such as agents, services, microservices, embedded systems, and so on, and what is more important, also by combining them all at the same time.

10.2 Future lines of research

The results presented in this document are the result of a continuously evolving body of work in many different contexts and with different objectives (EU-Provenance, IST-CONTRACT, ALIVE being the main ones, but with many side projects involved as well). At the same time, our focus has always been to try being as strict as possible from a formal perspective. Both aspects of our work have always been at conflict, and as such there are many improvements and lines of derived work that we propose:

- Convert the monitor into a cloud component capable of automatically distributing computational resources on split normative contexts, as well as implementing the capability of having dynamic normative contexts. This line of work has already been started by [Gómez-Sebastià and Alvarez-Napagao 2012; Gómez-Sebastià, Alvarez-Napagao, and Vázquez-Salceda 2011].
- Incorporate, at a formal level or implementation level or both, other social abstractions similar to norms but with different semantics, such as commitments or landmarks.
- Improve the norm lifecycle proposed in Section 6.4.2 with a formalism different from LTL that allows us to specify loops.
- Identify enforcement mechanisms and generalise, at a formal level, our operational semantics to allow reductions to them.
- Integrate our monitor not only on SOA governance architectures but also on any of the technologies currently adopted by microservices architectures, such as Event Sourcing or CQRS [Betts et al. 2013].
- Extend our use cases in domains with not so *apparent* needs for institutional abstractions, such as mobility or Internet of Things. Early encouraging attempts have already been made [Gómez-Sebastià, Alvarez-Napagao, and Vázquez-Salceda 2013; Gómez-Sebastià, Garcia-Gasulla, and Alvarez-Napagao 2011].

PART IV

APPENDICES

APPENDIX A

Proofs

This appendix provides the proofs for the deontic logic reductions of our normative framework described in Section 6.4. The purpose of these proofs is to demonstrate that our norm representation (a set of first-order logic formulas) can represent both deontic statements in Standard Deontic Logic and in Dyadic Logic.

A.1 Achievement obligations

In our framework, achievement obligations (typically, $O(A)$ where A is a state to be eventually achieved once in the future) are characterised by leaving the discharge condition as the unique free parameter:

$$\langle \pi, i, \theta \rangle \models \mathcal{O}_{\theta f_n^R \leq timeout}(E_\alpha[\theta\top] \preceq \theta f_n^D \mid \theta\top) \text{ iff } \langle \pi, i, \theta \rangle \models \exists j \geq i, \theta'' : \langle \pi, j, \theta\theta'' \rangle \models f_n^D$$

Proof of this equality is given in Section A.1.1. In the case of achievement obligations, axioms K (section A.1.2) and $Necessitation$ (section A.1.3) can be proven.

A relevant issue of achievement obligations in our framework is that axiom D is not fulfilled. The reason is that the negation of the reduction is simply $\neg f_n^D$, that is, that the state to be achieved is the complementary state of the original achievement obligation. The main implication of this is that, in our framework, when dealing with achievement obligations as commonly treated in SDL, we cannot ensure that $O(A)$ and $O(\neg A)$ are incompatible. While from a theoretical perspective this might seem a problem, from a practical point of view trying to ensure this property above others might be an even bigger problem: if the only thing we care about is to achieve two goal states and we do not care about maintenance, then it is not really a drawback if both obligations can be achieved at different points of time.

For simplification purposes in the subsequent proofs, we will assume that:

$$\mathcal{O}(f_n^D) \equiv \mathcal{O}_{\theta f_n^R \leq timeout}(E_\alpha[\theta\top] \preceq \theta f_n^D \mid \theta\top)$$

A.1.1 Substitution for achievement obligations

$$
\begin{array}{ll}
\quad\langle \pi, i, \theta \rangle \models & \mathcal{O}_{\theta f_n^R \leq timeout}(E_\alpha[\theta f_n^M] \preceq \theta f_n^D \mid \theta f_n^A) \\
\text{iff } \langle \pi, i, \theta \rangle \models & \mathbf{G}(\neg f_n^A) \vee \\
 & \qquad \Big[\neg f_n^A \mathbf{U} \Big(f_n^A \wedge [\forall \theta' : \theta' f_n^M \mathbf{U} \exists \theta'' : \theta'' f_n^D] \Big) \Big] \vee \\
 & \qquad \Big[\neg f_n^A \mathbf{U} \Big(f_n^A \wedge [\exists \theta' : \theta' f_n^M \mathbf{U} (\neg \theta' f_n^M \wedge [\exists \theta'' : \mathbf{F}_{\leq timeout} \theta'' \theta' f_n^R])] \Big) \Big] \\
\text{iff } \langle \pi, i, \theta \rangle \models & \mathbf{G}(\neg \top) \vee \\
 & \qquad \Big[\neg \top \mathbf{U} \Big(\top \wedge [\forall \theta' : \theta' \top \mathbf{U} \exists \theta'' : \theta'' f_n^D] \Big) \Big] \vee \\
 & \qquad \Big[\neg \top \mathbf{U} \Big(\top \wedge [\exists \theta' : \theta' \top \mathbf{U} (\neg \theta' \top \wedge [\exists \theta'' : \mathbf{F}_{\leq timeout} \theta'' \theta' \bot] \\
\text{iff } \langle \pi, i, \theta \rangle \models & \mathbf{G}(\bot) \vee \\
 & \qquad \Big[\mathbf{F}(\exists \theta'' : \theta'' f_n^D) \Big] \vee \\
 & \qquad \Big[\exists \theta' : \theta' \top \mathbf{U} (\bot \wedge [\bot]) \Big] \\
\text{iff } \langle \pi, i, \theta \rangle \models & \Big[\mathbf{F}(\exists \theta'' : \theta'' f_n^D) \Big] \\
\text{iff}_{Def\ 4.1.(g)} & \exists j \geq i, \theta'' : \langle \pi, i, \theta' \theta \rangle \models f_n^D
\end{array}
$$

The intuition behind this result is that in an achievement obligation, the only thing that matters is that the goal state is eventually achieved. In a norm of this kind the activating condition plays no role, as the obligation stands from the moment it is announced. There is also no maintenance to be done, as achieving the goal state does not depend on the previous states to the actual achievement.

A.1.2 Proof of K

Axiom K states that $\mathcal{O}(A \to B) \to (\mathcal{O}(A) \to \mathcal{O}(B))$.

$$
\begin{array}{l}
\langle \pi, i, \theta \rangle \models \mathcal{O}(A \to B) \\
\text{iff } \exists j \geq i, \theta' : \langle \pi, j, \theta' \theta \rangle \models (A \to B) \\
\text{iff } \exists j \geq i, \theta' : \langle \pi, j, \theta' \theta \rangle \models (\neg A \vee B) \\
\text{iff } \exists j \geq i, \theta' : \langle \pi, j, \theta' \theta \rangle \models \neg A \vee B \\
\text{iff } \exists j \geq i, \theta' : (\langle \pi, j, \theta' \theta \rangle \models \neg A \vee \langle \pi, j, \theta' \theta'' \rangle \models B) \\
\text{iff } \exists j \geq i, \theta' : (\langle \pi, j, \theta' \theta \rangle \models \neg A) \vee \langle \pi, j, \theta' \theta'' \rangle \models B) \\
\text{iff } \exists j \geq i, \theta' : (\langle \pi, j, \theta' \theta \rangle \not\models A \vee \langle \pi, j, \theta' \theta'' \rangle \models B) \\
\text{iff } (\exists j \geq i, \theta' : \langle \pi, j, \theta' \theta' \rangle \not\models A) \vee (\exists j \geq i, \theta' : \langle \pi, j, \theta' \theta \rangle \models B) \\
\text{iff } (\exists j \geq i, \theta' : \langle \pi, j, \theta' \theta \rangle \not\models A) \vee (\exists j \geq i, \theta' : \langle \pi, j, \theta' \theta \rangle \models B) \\
\text{iff } \langle \pi, i, \theta \rangle \models \neg \mathcal{O}(A) \vee \mathcal{O}(B) \\
\text{iff } \langle \pi, i, \theta \rangle \models \mathcal{O}(A) \to \mathcal{O}(B)
\end{array}
$$

Thus, in our framework, $\mathcal{O}(A \to B) \equiv \mathcal{O}(A) \to \mathcal{O}(B)$, so K is fulfilled.

A.1.3 Proof of Necessitation

$Necessitation$ states that $\models \alpha \to \langle \pi, i, \theta \rangle \models \mathcal{O}(\alpha)$: if something is a tautology, then it is obliged for it to happen.

$\models \alpha$
$\Rightarrow \forall \pi', i', \theta' : \langle \pi', i', \theta' \rangle \models \alpha$
$\Rightarrow_{\pi'=\pi} \forall i', \theta' : \langle \pi, i', \theta' \rangle \models \alpha$
$\Rightarrow \forall i', \exists \theta' : \langle \pi, i', \theta' \rangle \models \alpha$
$\Rightarrow \exists j \geq i, \theta' : \langle \pi, j, \theta' \rangle \models \alpha$
iff $\langle \pi, i, \theta \rangle \models \mathcal{O}(\alpha)$

A.2 Maintenance obligations

Maintenance obligations are seen in the literature in a very similar form to achievement obligations: $O(A)$, but in this case A is a state to be permanently maintained. In order to represent this in our framework, the maintenance condition has to be the unique free parameter:

$$\langle \pi, i, \theta \rangle \models \mathcal{O}_{\theta f_n^R \leq timeout}(E_\alpha[\theta f_n^M] \preceq \theta f_n^D \mid \theta f_n^A) \text{ iff } \langle \pi, i, \theta \rangle \models \forall \theta' : \mathbf{G}(\theta' f_n^M)$$

Proof of this equality is given in Section A.2.1. Interestingly, axioms K (section A.2.2), D (section A.2.3) and $Necessitation$ (section A.2.4) can be proven. Therefore, maintenance obligations in our framework are equivalent to maintenance obligations in SDL.

For simplification purposes in the subsequent proofs, we will assume that:

$$\mathcal{O}(f_n^M) \equiv \mathcal{O}_{\theta f_n^R \leq timeout}(E_\alpha[\theta f_n^M] \preceq \theta f_n^D \mid \theta f_n^A)$$

A.2.1 Substitution for maintenance obligations

$\langle \pi, i, \theta \rangle \models \mathcal{O}_{\theta f_n^R \leq timeout}(E_\alpha[\theta f_n^M] \preceq \theta f_n^D \mid \theta f_n^A)$
iff $\langle \pi, i, \theta \rangle \models \mathbf{G}(\neg f_n^A) \vee$
$\Big[\neg f_n^A \mathbf{U}\Big(f_n^A \wedge [\forall \theta' : \theta' f_n^M \mathbf{U} \exists \theta'' : \theta'' f_n^D]\Big)\Big] \vee$
$\Big[\neg f_n^A \mathbf{U}\Big(f_n^A \wedge [\exists \theta' : \theta' f_n^M \mathbf{U}(\neg \theta' f_n^M \wedge [\exists \theta'' : \mathbf{F}_{\leq timeout} \theta'' \theta' f_n^R])]\Big)\Big]$
iff $\langle \pi, i, \theta \rangle \models \mathbf{G}(\neg \top) \vee$
$\Big[\neg \top \mathbf{U}\Big(\top \wedge [\forall \theta' : \theta' f_n^M \mathbf{U} \exists \theta'' : \theta'' \infty]\Big)\Big] \vee$
$\Big[\neg \top \mathbf{U}\Big(\top \wedge [\exists \theta' : \theta' f_n^M \mathbf{U}(\neg \theta' f_n^M \wedge [\exists \theta'' : \mathbf{F}_{\leq timeout} \theta'' \theta' \bot])]\Big)\Big]$
iff $\langle \pi, i, \theta \rangle \models \mathbf{G}(\bot) \vee$
$\Big[[\forall \theta' : \theta' f_n^M \mathbf{U} \exists \theta'' : \theta'' \infty]\Big] \vee$
$\Big[[\exists \theta' : \theta' f_n^M \mathbf{U}(\neg \theta' f_n^M \wedge \bot)]\Big]$
iff $\langle \pi, i, \theta \rangle \models \Big[\forall \theta' : \theta' f_n^M \mathbf{U} \exists \theta'' : \theta'' \infty\Big]$
iff $\langle \pi, i, \theta \rangle \models \Big[\forall \theta' : \mathbf{G}(\theta' f_n^M)\Big]$
iff $\forall j \geq i, \theta' : \langle \pi, j, \theta' \theta \rangle \models f_n^M$

The intuition behind this result is that in a maintenance obligation, the only thing that matters is that the state is maintained until the end of time. In a norm of this kind the activating condition plays no role, as the obligation stands from the moment it is announced. There is also no discharge to be considered.

A.2.2 Proof of K

Axiom K states that $\mathcal{O}(A \to B) \to (\mathcal{O}(A) \to \mathcal{O}(B))$.

$\langle \pi, i, \theta \rangle \models [\mathcal{O}(A \to B) \wedge \mathcal{O}(A)]$
iff $\langle \pi, i, \theta \rangle \models [\forall \theta' : \mathbf{G}(\theta'(A \to B))] \wedge [\forall \theta' : \mathbf{G}(\theta' A)]$
iff $\langle \pi, i, \theta \rangle \models [\forall \theta' : \mathbf{G}(\theta'(A \to B)) \wedge \mathbf{G}(\theta' A)]$
iff $\langle \pi, i, \theta \rangle \models [\forall \theta' : \mathbf{G}(\theta'(A \to B \wedge A)]$
iff $\langle \pi, i, \theta \rangle \models [\forall \theta' : \mathbf{G}(\theta'(B)]$
iff $\langle \pi, i, \theta \rangle \models \mathcal{O}(B)$

The result is identical to the case of achievement obligations: $\mathcal{O}(A \to B) \equiv \mathcal{O}(A) \to \mathcal{O}(B)$, so K is fulfilled.

A.2.3 Proof of D

Axiom D states that $\mathcal{O}(A) \to \neg\mathcal{O}(\neg A)$.

$\langle \pi, i, \theta \rangle \models \mathcal{O}(A)$
iff $\langle \pi, i, \theta \rangle \models \forall \theta' : \mathbf{G}(\theta' A)$
$\Rightarrow \langle \pi, i, \theta \rangle \models \exists \theta' : \mathbf{G}(\theta' A)$
$\Rightarrow_{\mathbf{G}(A) \to \neg \mathbf{G}(\neg A)} \langle \pi, i, \theta \rangle \models \exists \theta' : \neg \mathbf{G}(\neg \theta' A)$
iff $\langle \pi, i, \theta \rangle \models \neg \forall \theta' : \mathbf{G}(\neg \theta' A)$
iff $\langle \pi, i, \theta \rangle \models \neg \forall \theta' : \mathbf{G}(\theta' \neg A)$
iff $\langle \pi, i, \theta \rangle \models \neg \Big[\forall \theta' : \mathbf{G}(\theta' \neg A)\Big]$
iff $\langle \pi, i, \theta \rangle \models \neg \mathcal{O}(\neg A)$

In achievement obligations, D cannot be proven, but this is not the case with maintenance obligations. This result is reasonable: because maintenance obligations require that states are fulfilled at all points of time, two obligations with complementary maintenance states are incompatible.

A.2.4 Proof of Necessitation

$Necessitation$ states that $\models \alpha \to \langle \pi, i, \theta \rangle \models \mathcal{O}(\alpha)$: if something is a tautology, then it is obliged for it to happen.

$\models \alpha$
$\Rightarrow \forall \pi', i', \theta' : \langle \pi', i', \theta' \rangle \models \alpha$
$\Rightarrow_{\pi' = \pi} \forall i', \theta' : \langle \pi, i', \theta' \rangle \models \alpha$

$\Rightarrow \forall i', \theta' : \langle \pi, i', \theta' \rangle \models \alpha$
$\Rightarrow \forall j \geq i, \theta' : \langle \pi, j, \theta' \rangle \models \alpha$
iff $\langle \pi, i, \theta \rangle \models \mathcal{O}(\alpha)$

A.3 Dyadic Deontic Logic

Dyadic Deontic Logic (DDL) [Prakken and Sergot 1997] is an extension of SDL that allows to model conditional obligations, in which the common form is $O(A|\ B)$, read as: "given that B, it is obliged that A", or otherwise, "in the context defined by the event B, it is obliged that A". It is not clear in the literature whether B triggers the context or is required at all times to maintain the context [Prakken and Sergot 1996], but it is of little importance in our case to define the reduction.

In the first case:

- $f_n^A \equiv B$
- $f_n^M \equiv A$
- $f_n^D \equiv G(f_n^M)$
- $f_n^R \equiv \bot$

And the reduction is (shown in A.3.1):

$$\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A\Big] \\ \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B$$

In the second case:

- $f_n^A \equiv B$
- $f_n^M \equiv (A \wedge B)$
- $f_n^D \equiv \neg B$
- $f_n^R \equiv \bot$

The reduction formula is not explored in this Section but is trivially achieved in an identical fashion:

$$\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee \\ (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \exists k : [\forall k > j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models (A \wedge B) \wedge \exists \theta' : \langle \pi, k, \theta'\theta \rangle \models \neg B]\Big] \\ \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B$$

In this annex we focus on the first case, but it is trivially provable that the axioms are equally valid on both systems. There exist several of such axiom systems that apply to DDL, but here we choose the $K1$-$K4$ of [Hilpinen 1971] and we prove $K1$, $K3$ and $K4$ in section A.3.2, A.3.4 and A.3.5.

Axiom $K2$, which is $\neg(\mathcal{O}(A|B) \wedge \mathcal{O}(\neg A|B))$, cannot be reduced to $\top$ in our framework, but to $\mathbf{F}(B)$, as seen in section A.3.3. This result, far from discouraging, is actually intuitive. If the condition never happens, can it be stated that it is not valid that

$\neg\mathcal{O}(A|B)$ and $\mathcal{O}(A|B)$ coexist? If we take as an example the extreme case: $B \equiv \bot$, is it really a contradiction? For us, it is not: complementary maintenance conditions are only inconsistent if it is a fact that the activating condition is eventually going to hold.

A.3.1 Substitution for dyadic deontic logic

$$\langle \pi, i, \theta \rangle \models \mathcal{O}_{\theta f_n^R \leq timeout}(E_\alpha[\theta f_n^M] \preceq \theta f_n^D \mid \theta f_n^A)$$

iff $\langle \pi, i, \theta \rangle \models \mathbf{G}(\neg f_n^A) \vee$
$\Big[\neg f_n^A \mathbf{U}\Big(f_n^A \wedge [\forall\theta' : \theta' f_n^M \mathbf{U} \exists\theta'' : \theta'' f_n^D]\Big)\Big] \vee$
$\Big[\neg f_n^A \mathbf{U}\Big(f_n^A \wedge [\exists\theta' : \theta' f_n^M \mathbf{U}(\neg\theta' f_n^M \wedge [\exists\theta'' : \mathbf{F}_{\leq timeout}\theta''\theta' f_n^R])]\Big)\Big]$

iff $\langle \pi, i, \theta \rangle \models \mathbf{G}(\neg f_n^A) \vee$
$\Big[\neg f_n^A \mathbf{U}\Big(f_n^A \wedge [\forall\theta' : \theta' f_n^M \mathbf{U} \exists\theta'' : \theta'' \mathbf{G}(\forall\theta' : \theta' f_n^M)]\Big)\Big] \vee$
$\Big[\neg f_n^A \mathbf{U}\Big(f_n^A \wedge [\exists\theta' : \theta' f_n^M \mathbf{U}(\neg\theta' f_n^M \wedge [\exists\theta'' : \mathbf{F}_{\leq timeout}\theta''\theta' \bot])]\Big)\Big]$

iff $\langle \pi, i, \theta \rangle \models \mathbf{G}(\neg f_n^A) \vee \Big[\neg f_n^A \mathbf{U}\Big(f_n^A \wedge \mathbf{G}(\forall\theta' : \theta' f_n^M)\Big)\Big]$

iff $\langle \pi, i, \theta \rangle \models (\neg f_n^A \mathbf{U}\mathbf{G}(\neg f_n^A)) \vee \Big[\neg f_n^A \mathbf{U}\Big(f_n^A \wedge \mathbf{G}(\forall\theta' : \theta' f_n^M)\Big)\Big]$

iff $\langle \pi, i, \theta \rangle \models \neg f_n^A \mathbf{U}\Big[\mathbf{G}(\neg f_n^A) \vee \Big(f_n^A \wedge \mathbf{G}(\forall\theta' : \theta' f_n^M)\Big)\Big]$

iff $\exists j \geq i : \langle \pi, j, \theta \rangle \models \Big[\mathbf{G}(\neg f_n^A) \vee \Big(f_n^A \wedge \mathbf{G}(\forall\theta' : \theta' f_n^M)\Big)\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg f_n^A$

iff $\Big[\exists j \geq i : \langle \pi, j, \theta \rangle \models \mathbf{G}(\neg f_n^A) \vee \exists j \geq i : \langle \pi, j, \theta \rangle \models \Big(f_n^A \wedge \mathbf{G}(\forall\theta' : \theta' f_n^M)\Big)\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg f$

iff $\Big[\forall j \ \geq i : \exists j \geq i : \langle \pi, j, \theta \rangle \models \neg f_n^A \vee$

$(\exists j \geq i : \langle \pi, j, \theta \rangle \models f_n^A \wedge \exists j \geq i : \langle \pi, j, \theta \rangle \models \mathbf{G}(\forall\theta' : \theta' f_n^M)\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg f_n^A$

iff $\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg f_n^A \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models f_n^A \wedge \forall j \geq i : \langle \pi, j, \theta \rangle \models (\forall\theta' : \theta' f_n^M)\Big]$
$\wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg f_n^A$

iff $\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg f_n^A \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models f_n^A \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models f_n^M)\Big]$
$\wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg f_n^A$

The resulting formula can be informally read as: "either the norm is never activated or else it is activated at some point for the first time and the maintenance condition is always fulfilled from that moment on".

A.3.2 Proof of K1

Axiom $K1$ states that $\mathcal{O}(A \vee \neg A|B)$, that is, that whichever the activating condition is, an obligation which contains both a maintenance condition and its complementary formula is a tautology.

$\langle \pi, i, \theta \rangle \models \mathcal{O}(A \vee \neg A|B)$

iff $\Big[\forall j \ \geq \ i \ : \ \langle \pi, j, \theta \rangle \ \models \ \neg B \vee (\exists j \ \geq \ i \ : \ \langle \pi, j, \theta \rangle \ \models \ B \wedge \forall j \ \geq \ i, \theta' \ : \ \langle \pi, j, \theta'\theta \rangle \ \models$

$(A \vee \neg A))\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B$

iff $\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models \top)\Big] \wedge \forall i \leq$

$k < j, \langle \pi, k, \theta \rangle \models \neg B$
iff $\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \top)\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B$

At this point we are left with two cases: the obligation gets activated or not. In both cases, the formula is a tautology:

If $\mathbf{G}(\neg B)$:
$\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \top)\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B$
iff $\Big[\top \vee (\bot \wedge \top)\Big] \wedge \top$
iff $\top$

If $\mathbf{F}(B)$:
$\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \top)\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B$
iff $\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\bot \wedge \top)\Big] \wedge \top$
iff $\top$

A.3.3 Proof of K2

Axiom $K2$ states that $\neg(\mathcal{O}(A|B) \wedge \mathcal{O}(\neg A|B))$, that is, that it cannot be the case that we have two obligations with the same activating condition and complementary maintenance obligations. This can be seen as the D axiom of SDL with $B \equiv \top$.

$\langle \pi, i, \theta \rangle \models \neg(\mathcal{O}(A|B) \wedge \mathcal{O}(\neg A|B))$
iff $\neg\Bigg[\Bigg(\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models$
$A)\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B\Bigg)$
$\wedge$
$\Bigg(\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models \neg A)\Big] \wedge \forall i \leq$
$k < j, \langle \pi, k, \theta \rangle \models \neg B\Bigg)\Bigg]$
iff $\neg\Bigg(\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models$
$A)\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B\Bigg)$
$\vee$
$\neg\Bigg(\Big[\forall j \ \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models \neg A)\Big] \wedge \forall i \leq$

$k < j, \langle \pi, k, \theta \rangle \models \neg B\Big)$

iff $\Big(\neg\Big[\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models$

$A)\Big] \vee \neg\forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B\Big)$

$\vee$

$\Big(\neg\Big[\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models$

$\neg A)\Big] \vee \neg\forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B\Big)$

iff $\Big(\Big[\neg\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \wedge \neg(\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models$

$A)\Big] \vee \exists i \leq k < j, \langle \pi, k, \theta \rangle \models B\Big)$

$\vee$

$\Big(\Big[\neg\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \wedge \neg(\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models$

$\neg A)\Big] \vee \exists i \leq k < j, \langle \pi, k, \theta \rangle \models B\Big)$

iff $\Big(\Big[\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge (\neg\exists j \geq i : \langle \pi, j, \theta \rangle \models B \vee \neg\forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models$

$A)\Big] \vee \exists i \leq k < j, \langle \pi, k, \theta \rangle \models B\Big)$

$\vee$

$\Big(\Big[\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge (\neg\exists j \geq i : \langle \pi, j, \theta \rangle \models B \vee \neg\forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models \neg A)\Big] \vee \exists i \leq$

$k < j, \langle \pi, k, \theta \rangle \models B\Big)$

iff $\Big(\Big[\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge (\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee \exists j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models$

$\neg A)\Big] \vee \exists i \leq k < j, \langle \pi, k, \theta \rangle \models B\Big)$

$\vee$

$\Big(\Big[\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge (\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee \exists j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A)\Big] \vee \exists i \leq$

$k < j, \langle \pi, k, \theta \rangle \models B\Big)$

iff $\exists i \leq k < j, \langle \pi, k, \theta \rangle \models B \vee \Bigg[\Bigg(\Big(\Big[\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge (\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee \exists j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models \neg A)\Big]\Big) \vee \Big(\Big[\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge (\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee \exists j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A)\Big]\Big)\Bigg]$

iff $\exists i \leq k < j, \langle \pi, k, \theta \rangle \models B \vee \exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \Bigg[\Big[(\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee \exists j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models \neg A)\Big] \vee \Big[(\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee \exists j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A)\Big]\Bigg]$

iff $\exists i \leq k < j, \langle \pi, k, \theta \rangle \models B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B)$
iff $\mathbf{F}(B) \vee \bot$
iff $\mathbf{F}(B)$

As seen earlier, it can be discussed whether this is a valid result.

A.3.4 Proof of K3

Axiom $K3$ states that $\mathcal{O}(A \wedge A'|B) \equiv (\mathcal{O}(A|B) \wedge \mathcal{O}(A'|B))$: DDL is closed under conjunction of the maintenance condition.

$\langle \pi, i, \theta \rangle \models \mathcal{O}(A \wedge A'|B)$

iff $\Big[\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models (A \wedge A'))\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B$

iff $\Big[\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A')\Big] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B$

iff $\Bigg[\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee \Big[(\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A) \wedge (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A')\Big]\Bigg] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B$

iff $\Bigg[\Big[\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \wedge (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A)\Big] \wedge \Big[\forall j \geq i : \langle \pi, j, \theta \rangle \models \neg B \wedge (\exists j \geq i : \langle \pi, j, \theta \rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A')\Big]\Bigg] \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B$

iff $\Big[\big[\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B \wedge (\exists j \geq i : \langle \pi, j, \theta\rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A)\big] \wedge \forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg B\Big] \wedge \Big[\big[\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B \wedge (\exists j \geq i : \langle \pi, j, \theta\rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A')\big] \wedge \forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg B\Big]$

iff $\langle \pi, i, \theta\rangle \models \mathcal{O}(A|B) \wedge \langle \pi, i, \theta\rangle \models \mathcal{O}(A'|B)$

This is a straightforward proof that is trivially achieved by the properties of FO-LTL.

A.3.5 Proof of K4

Axiom $K4$ states that $\mathcal{O}(A|B \vee B') \equiv (\mathcal{O}(A|B) \wedge \mathcal{O}(A|B'))$, that is, DDL is closed under disjunction of the activating condition.

$\langle \pi, i, \theta\rangle \models \mathcal{O}(A \wedge A'|B)$

iff $\big[\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg(B \vee B') \vee (\exists j \geq i : \langle \pi, j, \theta\rangle \models (B \vee B') \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A)\big] \wedge \forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg(B \vee B')$

iff $\big[\forall j \geq i : \langle \pi, j, \theta\rangle \models (\neg B \wedge \neg B')) \vee (\exists j \geq i : \langle \pi, j, \theta\rangle \models (B \vee B') \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A)\big] \wedge \forall i \leq k < j, \langle \pi, k, \theta\rangle \models (\neg B \wedge \neg B')$

iff $\big[(\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B \wedge \forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B') \vee [(\exists j \geq i : \langle \pi, j, \theta\rangle \models B \vee \exists j \geq i : \langle \pi, j, \theta\rangle \models B'] \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A)\big] \wedge (\forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg B \wedge \forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg B')$

iff $\big[(\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B \wedge \forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B') \vee [(\exists j \geq i : \langle \pi, j, \theta\rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A) \vee (\exists j \geq i : \langle \pi, j, \theta\rangle \models B' \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A)]\big] \wedge (\forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg B \wedge \forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg B')$

iff $\Big[\big(\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B \vee (\exists j \geq i : \langle \pi, j, \theta\rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A) \vee (\exists j \geq i : \langle \pi, j, \theta\rangle \models B' \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A)\big) \wedge \big(\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B' \vee (\exists j \geq i : \langle \pi, j, \theta\rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A) \vee (\exists j \geq i : \langle \pi, j, \theta\rangle \models B' \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A)\big)\Big] \wedge (\forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg B \wedge \forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg B')$

iff $\Big[\big(\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B \vee \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A \vee (\exists j \geq i : \langle \pi, j, \theta\rangle \models B' \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A)\big) \wedge \big(\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B' \vee (\exists j \geq i : \langle \pi, j, \theta\rangle \models B \wedge \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A) \vee \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A\big)\Big] \wedge (\forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg B \wedge \forall i \leq k < j, \langle \pi, k, \theta\rangle \models \neg B')$

iff $\Big[\big(\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B \vee \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta\rangle \models A\big) \wedge \big(\forall j \geq i : \langle \pi, j, \theta\rangle \models \neg B' \vee$

$\forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A \big) \big] \wedge (\forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B')$
iff $\Big[\Big(\forall j \; \geq i : \langle \pi, j, \theta \rangle \models \neg B \vee \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A \Big) \wedge (\forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B \Big] \wedge$
$\Big[\Big(\forall j \; \geq i : \langle \pi, j, \theta \rangle \models \neg B' \vee \forall j \geq i, \theta' : \langle \pi, j, \theta'\theta \rangle \models A \Big) \wedge \forall i \leq k < j, \langle \pi, k, \theta \rangle \models \neg B') \Big]$
iff $\langle \pi, i, \theta \rangle \models \mathcal{O}(A|\; B) \wedge \langle \pi, i, \theta \rangle \models \mathcal{O}(A|\; B')$

Similarly to what happens with the previous axiom, this is also a trivial result based on the properties of FO-LTL.

Bibliography

Abrahams, Alan S and Bacon, Jean M (2002). “The life and times of identified, situated, and conflicting norms”. In: *Sixth International Workshop on Deontic Logic in Computer Science (DEON)*, pp. 3–20.

Agents, Foundation for Intelligent Physical (2000). “FIPA SL Content Language Specification”. In: *Foundation for Intelligent Physical Agents*. URL: http://standards.computer.org/fipa/specs/fipa00008/XC00008F.html.

— (2002a). “Fipa ACL Message Structure Specification”. In: *Foundation for Intelligent Physical Agents, Geneva, Switzerland*. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=11355975404632573812related:dIcyx3aHmJ0J.

— (2002b). “FIPA ACL Message Structure Specification”. In: *Foundation for Intelligent Physical Agents*. URL: http://standards.computer.org/fipa/specs/fipa00061/XC00061D.html.

— (2003). “FIPA Interaction Protocol Library Specification”. In: pp. 1–25.

Ågotnes, Thomas; Hoek, Wiebe van der and Wooldridge, Michael (2010). “Robust normative systems and a logic of norm compliance”. English. In: *Logic Journal of IGPL* 18.1, pp. 4–30. DOI: 10.1093/jigpal/jzp070. URL: http://jigpal.oxfordjournals.org/cgi/doi/10.1093/jigpal/jzp070.

Alberola, Juan M.; Such, José M.; Espinosa, Agustín; Botti, Vicente and García-Fornés, Ana (2008). *Magentix: a multiagent platform integrated in Linux*. Bath, UK: Proceedings of the Sixth European Workshop on Multi-Agent Systems (EUMAS-2008). URL: http://scholar.google.com/scholar?q=related:ttStxEmuGScJ:scholar.google.com/&hl=en&num=20&as_sdt=0,5.

Albert, Luc and Fages, François (1988). “Average Case Complexity Analysis of the Rete Multi-Pattern Match Algorithm”. English. In: *ICALP* 317.Chapter 2, pp. 18–37. DOI: 10.1007/3-540-19488-6_104. URL: http://dx.doi.org/10.1007/3-540-19488-6_104.

Aldewereld, Huib (2007). “Autonomy vs. Conformity: An Institutional Perspective on Norms and Protocols”. In: *PhD Thesis, Utrecht University*. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=1422272135052886312.

Aldewereld, Huib; Dignum, Frank and Meyer, John-Jules Ch (2007). “Designing protocols for agent institutions”. English. In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems* 66.4, pp. 1173–1179. URL: http://

www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=7636791128828226120related:SCqp-RFV-2kJ.

Aldewereld, Huib and Dignum, Virginia (2010a). "OperettA: Organization-Oriented Development Environment". English. In: *Languages, Methodologies, and Development Tools for Multi-Agent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–18. ISBN: 978-3-642-22722-6. DOI: 10.1007/978-3-642-22723-3_1. URL: http://link.springer.com/10.1007/978-3-642-22723-3_1.

— (2010b). "OperettA: organization-oriented development environment". In: *LADS'10: Proceedings of the Third international conference on Languages, methodologies, and development tools for multi-agent systems*. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=2032683.2032684&coll=DL&dl=GUIDE&CFID=546795349&CFTOKEN=44049066.

Aldewereld, Huib; Grossi, Davide; Vázquez-Salceda, Javier and Dignum, Frank (2005). "Designing Normative Behaviour by the Use of Landmarks". In: *Proceedings of AAMAS-05 International Workshop on Agents, Norms and Institutions for Regulated Multiagent Systems*. URL: http://www.cs.uu.nl/~huib/downloads/anirem05-landmarks.PDF.

Aldewereld, Huib; Padget, Julian; Vasconcelos, Wamberto; Vázquez-Salceda, Javier; Sergeant, Paul and Staikopoulos, Athanasios (2010a). "Adaptable, Organization-Aware, Service-Oriented Computing". In: *Intelligent Systems* 25.4, pp. 80–84. DOI: 10.1109/MIS.2010.93. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5552590.

Aldewereld, Huib; Alvarez-Napagao, Sergio; Dignum, Frank and Vázquez-Salceda, Javier (2010b). "Making norms concrete". In: *AAMAS '10: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems* Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, pp. 807–814. URL: http://dl.acm.org/citation.cfm?id=1838206.1838314.

Alvarez-Napagao, Sergio; Cliffe, Owen; Vázquez-Salceda, Javier and Padget, Julian (2009). "Norms, organisations and semantic web services: The ALIVE approach". English. In: Coordination, Organization, Institutions and Norms in Agent Systems & On-line Communities (COIN@MALLOW'009), Proceedings of the Second Multi-Agent Logics, Languages, and Organisations Federated Workshops, Volume 494, pp. 1–2. ISSN: 1613-0073. URL: http://upcommons.upc.edu//handle/2117/14283.

Alvarez-Napagao, Sergio; Aldewereld, Huib; Vázquez-Salceda, Javier and Dignum, Frank (2011). "Normative Monitoring: Semantics and Implementation". English. In: *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 321–336. ISBN: 978-3-642-21267-3. DOI: 10.1007/978-3-642-21268-0_18. URL: http://link.springer.com/10.1007/978-3-642-21268-0_18.

Alvarez-Napagao, Sergio; Gómez-Sebastià, Ignasi; Panagiotidi, Sofia; Tejeda-Gómez, Arturo; Oliva-Felipe, Luis and Vázquez-Salceda, Javier (2012). "Socially-Aware Emergent Narrative". English. In: *Lecture Notes in Computer Science*. Ed. by Martin Beer; Cyril Brom; Frank Dignum and Von-Wun Soo. Berlin, Heidelberg: Springer

Berlin Heidelberg, pp. 139–150. ISBN: 978-3-642-32325-6. DOI: 10.1007/978-3-642-32326-3_9. URL: http://www.lsi.upc.edu/~igomez/Papers/AEGS_2011.pdf.

Andrieux, Alain; Czajkowski, Karl; Dan, Asit; Keahey, Kate; Ludwig, Heiko; Nakata, Toshiyuki; Pruyne, Jim; Rofrano, John; Tuecke, Steve and Xu, Ming (2005). "Web Services Agreement Specification (WS-Agreement) Version 2005/09". In: *Global Grid Forum*. URL: http://xml.coverpages.org/WS-Agreement-13652.pdf.

Annichiarico, Roberta and Cortés, Ulises (2010). "To Share or Not to Share SHARE-it: Lessons Learnt". In: *eHealth*, pp. 295–302.

Arnold, Ken (1999). "The Jini architecture: dynamic services in a flexible network". In: *36th Annual Conference on Design Automation (DAC'99)*, pp. 157–162. DOI: 10.1109/DAC.1999.168. URL: http://doi.ieeecomputersociety.org/10.1109/DAC.1999.168.

Artikis, Alexander; Sergot, Marek and Pitt, Jeremy (2009). "Specifying norm-governed computational societies". English. In: *ACM Transactions on Computational Logic (TOCL)* 10.1, pp. 1–42. DOI: 10.1145/1459010.1459011. URL: http://portal.acm.org/citation.cfm?doid=1459010.1459011.

Austin, John Langshaw (1975). *How to Do Things with Words*. English. Harvard University Press. ISBN: 9780674411524. URL: http://scholar.google.com/scholar?q=related:QL0sbUP7lTQJ:scholar.google.com/&hl=en&num=30&as_sdt=0,5&as_ylo=1962&as_yhi=1962.

Banerji, Arindam; Bartolini, Claudio; Beringer, Dorothea; Chopella, Venkatesh; Govindarajan, Kannan; Karp, Alan; Kuno, Harumi; Lemon, Mike; Pogossiants, Gregory; Sharma, Shamik and Williams, Scott (2002). "Web Services Conversation Language (WSCL) 1.0". In: *W3C Note*. URL: http://www.w3.org/TR/2002/NOTE-wscl10-20020314.

Bargh, John A; Gollwitzer, Peter M; Lee-Chai, Annette; Barndollar, Kimberly and Trötschel, Roman (2001). "The automated will: Nonconscious activation and pursuit of behavioral goals." English. In: *Journal of Personality and Social Psychology* 81.6, pp. 1014–1027. DOI: 10.1037/0022-3514.81.6.1014. URL: http://psycnet.apa.org/journals/psp/81/6/1014.html.

Bellifemine, Fabio; Poggi, Agostino and Rimassa, Giovanni (2001). "Developing multi-agent systems with JADE". In: *INTELLIGENT AGENTS VII AGENT THEORIES ARCHITECTURES AND LANGUAGES*. URL: http://www.springerlink.com/index/1RLAKJUC8243RTME.pdf.

Bellwood, Thomas A (2001). "UDDI-A Foundation for Web Services". In: *Proceedings of XML Conference & Exposition. Orlando*. URL: http://www.idealliance.org/papers/xml2001/papers/pdf/03-02-03.pdf.

Betts, Dominic; Dominguez, Julian; Melnik, Grigori; Simonazzi, Fernando and Subramanian, Mani (2013). *Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure*. Microsoft patterns & practices. ISBN: 1621140164. URL: http://dl.acm.org/citation.cfm?id=2509680.

Bicchieri, Cristina (2006). *The Grammar of Society*. English. The Nature and Dynamics of Social Norms. Cambridge University Press. ISBN: 9780521573726. URL: http://

books.google.com/books?hl=en&lr=&id=4N1FDIZvcI8C&oi=fnd&pg=PR9&dq=cristina+bicchieri&ots=TddVWPnHRr&sig=c9jK3vUppXfdYI9gUrP-kyeeh70.

Bloomberg, Jason (2013). *The Agile Architecture Revolution*. John Wiley & Sons. URL: https://books.google.co.uk/books/about/The_Agile_Architecture_Revolution.html?id=Y4JkFk6lxOcC.

Boella, Guido and Torre, Leendert van der (2004). “Regulative and constitutive norms in normative multiagent systems”. In: *Procs. of KR'04*, pp. 255–265. URL: http://www.aaai.org/Papers/KR/2004/KR04-028.pdf.

Boissier, Olivier; Bordini, Rafael H; Hübner, Jomi F; Ricci, Alessandro and Santi, Andrea (2013). “Multi-agent oriented programming with JaCaMo”. English. In: *Science of Computer Programming* 78.6, pp. 747–761. DOI: 10.1016/j.scico.2011.10.004. URL: http://linkinghub.elsevier.com/retrieve/pii/S016764231100181X.

Boley, Harold (2006). “The RuleML Family of Web Rule Languages”. In: *LECTURE NOTES IN COMPUTER SCIENCE*. URL: http://www.springerlink.com/index/18ng4244p7216j51.pdf.

Booth, David; Haas, Hugo; McCabe, Francis; Newcomer, Eric; Champion, Michael; Ferris, Christopher and Orchard, David (2004). “Web Services Architecture”. In: URL: http://www.w3.org/TR/ws-arch/wsa.pdf.

Bordini, Rafael H and Hübner, Jomi F (2005). “BDI Agent Programming in AgentSpeak Using Jason”. English. In: *Computational Logic in Multi-Agent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 143–164. ISBN: 978-3-540-33996-0. DOI: 10.1007/11750734_9. URL: http://link.springer.com/10.1007/11750734_9.

— (2006). “BDI agent programming in AgentSpeak using Jason”. In: *Computational logic in multi-agent systems*. URL: http://www.springerlink.com/index/r3012556l13675k3.pdf.

Bougiouklis, Kostas (2008). “PANDA: Collaborative Process Automation Support using Service Level Agreements and Intelligent dynamic Agents in SME clusters”. In: p. 6.

Cardoso, Henrique and Oliveira, Eugenio (2010). “Directed Deadline Obligations in Agent-Based Business Contracts”. In: *COIN V*. Ed. by Julian et al Padget. Springer Berlin / Heidelberg, pp. 225–240. ISBN: 978-3-642-14961-0. URL: http://www.worldcat.org/title/coordination-organizations-institutions-and-norms-in-agent-systems-v-coin-2009-international-workshops-coinaamas-2009-budapest-hungary-may-2009-coinijcai-2009-pasadena-usa-july-2009-coinmallow-2009-turin-italy-september-2009-revised-selected-papers/oclc/668095878.

Cardoso, Henrique Lopes and Oliveira, Eugenio (2000). “Using and Evaluating Adaptive Agents for Electronic Commerce Negotiation”. In: *Proceedings of the International Joint Conference*. URL: http://portal.acm.org/citation.cfm?id=645852.669324.

— (2009). “A Context-Based Institutional Normative Environment”. In: *Coordination, Organizations, Institutions and Norms in Agent Systems IV*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 140–155. ISBN: 978-3-642-00442-1. DOI: 10.1007/978-3-642-00443-8_10. URL: http://www.springerlink.com/index/01P737668774024M.pdf.

Charles, Darryl (2003). “Enhancing gameplay: Challenges for artificial intelligence in digital games”. In: *Proceedings of the 1st World Conference on Digital Games Research Conference*. Utrecht, The Netherlands, p. 10. URL: http://www.researchgate.net/profile/Darryl_Charles/publication/221217607_Enhancing_gameplay_challenges_for_articifical_intelligence_in_digital_games/links/02e7e52af5b85d1b15000000.pdf.

Cirstea, Horatiu; Kirchner, Claude; Moossen, Michael and Moreau, Pierre-Etiene (2008). “Production Systems and Rete Algorithm Formalisation”. In: *techreport* ILOG, INRIA Lorraine, INRIA Rocquencourt.

Clement, Luc (2005). “Risks of Running SOA without Registry”. In: *Loosely Coupled*. URL: http://www.trainingbyroi.com/Java%20Info/SOAwithoutRegistryRisky.pdf.

Coles, Amanda; Coles, Andrew; Olaya, Angel García; Jiménez, Sergio; López, Carlos Linares; Sanner, Scott and Yoon, Sungwook (2012). “A Survey of the Seventh International Planning Competition”. English. In: *AI Magazine* 33.1, pp. 83–88. DOI: 10.1609/aimag.v33i1.2392. URL: http://www.aaai.org/ojs/index.php/aimagazine/article/view/2392.

Colley, John L; Doyle, Jacqueline L; Stettinius, Wallace and Logan, George (2003). “Corporate Governance”. In: *McGraw-Hill Professional*. URL: http://books.google.com/books?hl=en&lr=&ie=UTF-8&id=WnK2o1GvEt0C&oi=fnd&pg=PP18&dq=corporate+governance+concepts+colley&ots=MccldmPn22&sig=riStkpBRFo_TJPjrDHsIEBojrj8.

Community, JBoss. *JBoss Drools Business Rules, http://www.jboss.org/drools*. URL: http://www.jboss.org/drools/.

Condry, M; Gall, U and Delisle, P (1999). “Open Service Gateway architecture overview”. In: *Industrial Electronics Society* 2, pp. 735–742. DOI: 10.1109/IECON.1999.816492. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=816492.

Cortés, Ulises and Rodríguez-Aguilar, Juan Antonio (2000). “Trading agents in auction-based tournaments”. In: *Special issue on Intelligent Agents, INFORMATIQUE 1/2000:39-50*. URL: http://www.lsi.upc.edu/~ia/agentes/a001Rodriguez.pdf.

Criado, Natalia; Argente, Estefania; Noriega, Pablo and Botti, Vicente (2010). “Towards a Normative BDI Architecture for Norm Compliance”. In: *COIN@ MALLOW2010*, pp. 1–16. URL: http://ai-lab-webserver.aegean.gr/coin@mallow2010/COIN@MALLOW_pre-proceedings.pdf#page=9.

Curbera, F; Duftler, M; Khalaf, R; Nagy, W; Mukhi, N and Weerawarana, S (2002). “Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI”. English. In: *IEEE Internet Computing* 6.2, pp. 86–93. DOI: 10.1109/4236.991449. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=991449.

Dastani, Mehdi (2008). “2APL: a practical agent programming language”. English. In: *Autonomous Agents and Multi-Agent Systems* 16.3, pp. 214–248. DOI: 10.1007/s10458-008-9036-y. URL: http://link.springer.com/10.1007/s10458-008-9036-y.

Davidson, Donald (1971). *Agency*. Essays on actions and events. URL: http://scholar.google.com/scholar?q=related:0cLdkXttDiYJ:scholar.google.com/&hl=en&num=20&as_sdt=0,5&as_ylo=1971&as_yhi=1971.

Davidson, Donald (2002). “Mental Events”. In: *Contemporary Materialism*. Ed. by Paul K Moser and J D Trout. Contemporary materialism: a reader. URL: https://books.google.co.uk/books/about/Contemporary_Materialism.html?id=StyJAgAAQBAJ.

Davis, Randall and King, Jonathan (1975). “An overview of production systems”. In: *techreport*. URL: http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA019702.

De Vos, Marina; Padget, Julian and Satoh, Ken (2010). “Legal Modelling and Reasoning Using Institutions”. English. In: *New Frontiers in Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 129–140. ISBN: 978-3-642-25654-7. DOI: 10.1007/978-3-642-25655-4_12. URL: http://link.springer.com/10.1007/978-3-642-25655-4_12.

Dellarocas, Chrysanthos (2000). “Contractual Agent Societies: Negotiated shared context and social control in open multi-agent systems”. In: *Social Order in Multiagent Systems, Springer, ISBN 0792374509*. URL: http://ccs.mit.edu/dell/aa2000/paper13.pdf.

Dennett, Daniel Clement (1989). *The Intentional Stance*. URL: https://books.google.co.uk/books/about/The_Intentional_Stance.html?id=Qbvkja-J9iQC.

Dietrich, Jens (2005). “The Mandarax Manual”. In: *Available at: http://fisheye.cenqua.com*. URL: http://fisheye.cenqua.com/viewrep/~raw,r=1.8/mandarax/mandarax/docs/manual.pdf.

Dignum, Frank (2008). “On-line Adapting Games using Agent Organizations”. In: *IEEE Symposium on Computational Intelligence and Games (CIG’08)*, pp. 243–250. DOI: 978-1-4244-2974-5/08. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=related:6oqiVyAd9d4J.

Dignum, Frank and Vázquez-Salceda, Javier (2005). “Omni: Introducing social structure, norms and ontologies into agent organizations”. In: *Programming Multi-Agent Systems* 3346.Chapter 10, pp. 181–198. DOI: 10.1007/978-3-540-32260-3_10. URL: http://link.springer.com/10.1007/978-3-540-32260-3_10.

Dignum, Frank; Broersen, Jan; Dignum, Virginia and Meyer, John-Jules Ch (2004). “Meeting the Deadline: Why, When and How”. In: *Formal Approaches to Agent-Based Systems*. Ed. by James L Rash; Walter F Truszkowski and Christopher A Rouff. Berlin Heidelberg: Springer, pp. 30–40.

Dignum, Frank; Dignum, Virginia; Thangarajah, John; Padgham, Lin and Winikoff, Michael (2007). “Open Agent Systems ???” English. In: *Agent-Oriented Software Engineering VIII*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 73–87. ISBN: 978-3-540-79487-5. DOI: 10.1007/978-3-540-79488-2_6. URL: http://link.springer.com/10.1007/978-3-540-79488-2_6.

Dignum, Virginia (2004). “A Model for Organizational Interaction: Based on Agents, Founded in Logic”. In: *PhD Thesis, Utrecht University* 1, pp. 1–284. URL: http://igitur-archive.library.uu.nl/dissertations/2003-1218-115420/UUindex.html.

Dignum, Virginia; Weigand, Hans and Xu, L (2002). “Agent Societies: Towards Frameworks-Based Design”. In: *LECTURE NOTES IN COMPUTER SCIENCE*. URL: http://www.springerlink.com/index/427GFV0D6KJ400LE.pdf.

Dignum, Virginia; Meyer, John-Jules Ch; Weigand, Hans and Dignum, Frank (2008). "An Organization-oriented Model for Agent Societies". In: *Proceedings of International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02), at AAMAS, Bologna, Italy*, p. 20.

d'Inverno, Mark and Luck, Michael (2004). "Understanding Agent Systems". In: *Springer*. URL: http://books.google.com/books?hl=en&lr=&ie=UTF-8&id=GQcXCWErPZoC&oi=fnd&pg=PR3&dq=Understanding+Agent+Systems&ots=DhQq2uKyEz&sig=PFv6WeeUU5tc2FrOCzCOv_OvTjI.

Economic Social Affairs, Population Division UN Department of (2004). *World population to 2300*. Tech. rep. URL: http://www.un.org/esa/population/publications/longrange2/WorldPop2300final.pdf.

— (2012). *Population ageing and development: Ten years after Madrid*. Tech. rep. 2012/4. URL: http://www.un.org/esa/population/publications/popfacts/popfacts_2012-4.pdf.

Eijk, Rogier M van; Huget, Marc-Philippe and Dignum, Frank (2005). ***Agent Communication:*** *International Workshop on Agent Communication*. Springer. URL: https://books.google.es/books/about/Agent_Communication.html?hl=es&id=g-kFCAAAQBAJ.

Elster, J (1989). "Social Norms and Economic Theory". In: *The Journal of Economic Perspectives* 3.4, pp. 99–117. DOI: 10.2307/1942912. URL: http://www.jstor.org/stable/1942912.

Erl, Thomas (2004). "Service-oriented architecture". In: *Prentice Hall*. URL: http://www.serviceoriented.ws/Erl_SOABook1_Ch07-2.pdf.

Esteva, M; Rodriguez-Aguilar, J A; Arcos, J LL; Sierra, C; Noriega, Pablo; Rosell, B and Cruz, D de la (2008). "Electronic institutions development environment". In: *AAMAS '08 Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers*, pp. 1657–1658. URL: http://dl.acm.org/citation.cfm?id=1402744.1402751.

Esteva, Marc; Padget, Julian and Sierra, Carles (2002). "Formalizing a language for institutions and norms". In: *Intelligent Agents VIII*. URL: http://www.springerlink.com/index/edw0x9um0dpxbh8j.pdf.

Faci, N; Modgil, S; Oren, N; Meneguzzi, Felipe and Miles, S (2008). "Towards a monitoring framework for agent-based contract systems". In: *Cooperative Information Agents XII* 5180.Chapter 23, pp. 292–305. DOI: 10.1007/978-3-540-85834-8_23. URL: http://link.springer.com/10.1007/978-3-540-85834-8_23.

Farrell, Andrew D H; Sergot, Marek J; Sallé, Mathias and Bartolini, Claudio (2005). "Using the Event Calculus for Tracking the Normative State of Contracts". English. In: *International Journal of Cooperative Information Systems* 14.02n03, pp. 99–129. DOI: 10.1142/S0218843005001110. URL: http://www.hpl.hp.com/personal/Claudio_Bartolini/download/IJCIS_Farrell_Sergot.pdf.

Finin, Tim; Fritzson, R; McKay, D and McEntire, R (1994). "KQML as an agent communication language". In: *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), ACM Press*. URL: http://portal.acm.org/citation.cfm?id=191246.191322.

Finkelstein, Lawrence S (1995). “What Is Global Governance”. In: *Global Governance*. URL: http://www.agropolis.fr/formation/dd/nov04/what_is_global_governance.pdf.

Fornara, Nicoletta and Colombetti, Marco (2009). “Specifying and Enforcing Norms in Artificial Institutions”. In: *In M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, Springer-Verlag Berlin Heidelberg*, pp. 1–17. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=14954202287096801360related:UCi91b__h88J.

Foster, Ian; Jennings, Nicholas R and Kesselman, Carl (2004). “Brain meets brawn: why grid and agents need each other”. In: *3rd International Conference on Autonomous Agents and Multi-Agent Systems*. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1373456.

Fowler, M and Lewis, J (2014). *Microservices*. ThoughtWorks. URL: http://scholar.google.com/scholar?q=related:FC2s8-5Y57oJ:scholar.google.com/&hl=en&num=20&as_sdt=0,5.

Fox, Maria and Long, Derek (2009). “PDDL 2.1 : An Extension to PDDL for Expressing Temporal Planning Domains”. In: *University of Durham, UK*, pp. 1–48.

Fulton, Larry; Heffner, Randy and D’Silva, David (2008). “The Forrester Wave: SOA Service Life-Cycle Management, Q1 2008”. In: *Forrester Research*.

Garcia, Pere; Giménez, Eduard; Godo, Lluis and Rodríguez-Aguilar, Juan Antonio (1999). “Bidding Strategies for Trading Agents in Auction-Based Tournaments”. In: *AMET-98, Lecture Notes in Computer Science, LNAI 1571*, pp. 151–165. URL: http://www.springerlink.com/index/XH6RAL3LX37JRCH4.pdf.

García-Camino, A; Rodríguez-Aguilar, J; Sierra, Carles and Vasconcelos, Wamberto (2006). “A rule-based approach to norm-oriented programming of electronic institutions”. English. In: *ACM SIGecom Exchanges* 5.5, pp. 33–40. DOI: 10.1145/1124566.1124571. URL: http://portal.acm.org/citation.cfm?doid=1124566.1124571.

García-Camino, Andrés; Noriega, Pablo and Rodríguez-Aguilar, Juan Antonio (2005). “Implementing norms in electronic institutions”. In: *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems* Utrecht, Netherlands, pp. 667–673. URL: http://portal.acm.org/citation.cfm?id=1082473.1082575.

García-Camino, Andrés; Rodríguez-Aguilar, Juan A; Sierra, Carles and Vasconcelos, Wamberto (2009). “Constraint rule-based programming of norms for electronic institutions”. In: *Autonomous Agents and Multi-Agent Systems (Springer US)* 18.1, pp. 186–217. URL: http://www.springerlink.com/index/3H4104073465N082.pdf.

Genesereth, M and Fikes, Richard (1992). “Knowledge interchange format (KIF), version 3.0 reference manual”. In: *Tech. Rep. KSL-92-86*.

Gerevini, Alfonso and Long, Derek (2005). *Plan constraints and preferences in PDDL3: The Language of the Fifth International Planning Competition*. Tech. rep. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=15869234077132803629related:LU4GeVDYOtwJ.

Gómez-Sebastià, Ignasi (2016). “NoMoDEI: A framework for Norm Monitoring on Dynamic Electronic Institutions”. PhD thesis. Barcelona, Spain (January 2016).

Gómez-Sebastià, Ignasi and Alvarez-Napagao, Sergio (2012). "Towards Runtime Support for Norm Change from a Monitoring Perspective". In: Proceedings of the First International Conference on Agreement Technologies, Volume 918. URL: http://vi.ikt.ui.sav.sk/@api/deki/files/2119/=example.pdf.

Gómez-Sebastià, Ignasi; Alvarez-Napagao, Sergio and Vázquez-Salceda, Javier (2011). "A Distributed Norm Compliance Model". In: *Frontiers in Artificial Intelligence and Applications* Volume 232: Artificial Intelligence Research and Development, pp. 110–119. DOI: 10.3233/978-1-60750-842-7-110. URL: http://ebooks.iospress.nl/publication/6573.

— (2013). "Towards Heuristic Based Mobility Policy Optimisation". In: *Frontiers in Artificial Intelligence and Applications* Volume 256: Artificial Intelligence Research and Development, pp. 297–300. DOI: 10.3233/978-1-61499-320-9-297. URL: http://ebooks.iospress.nl/publication/35263.

Gómez-Sebastià, Ignasi; Garcia-Gasulla, Dario and Alvarez-Napagao, Sergio (2011). "Society of situated agents for adaptable eldercare". In: *ERCIM News* 87, pp. 23–24. URL: http://ercim-news.ercim.eu/en87/special/a-society-of-situated-agents-for-adaptable-eldercare.

Gómez-Sebastià, Ignasi; Garcia-Gasulla, Dario; Alvarez-Napagao, Sergio; Vázquez-Salceda, Javier and Cortés, Ulises (2012). "Towards an implementation of a social electronic reminder for pills". In: *VII Workshop on Agents Applied inHealth Care, AHC@AAMAS2012*. Valencia, Spain. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.466.1786&rep=rep1&type=pdf#page=63.

Gómez-Sebastià, Ignasi; Alvarez-Napagao, Sergio; Garcia-Gasulla, Dario and Cortés, Ulises (2013). "Situated agents and humans in social interaction for elderly healthcare: the case of COAALAS ". In: *14th Conference on Artificial Intelligence in Medicine* Proceedings of the VIII Workshop on Agents Applied in Health Care (A2HC2013), pp. 105–119. URL: http://upcommons.upc.edu/e-prints/bitstream/2117/23299/1/Situated%20agents%20and%20humans%20in%20social%20interaction%20for%20elderly%20healthcare:The%20case%20of%20COAALAS.pdf.

Gómez-Sebastià, Ignasi; Moreno, Jonathan; Alvarez-Napagao, Sergio; Garcia-Gasulla, Dario; Barrué, Cristian and Cortés, Ulises (2015). "Situated agents and humans in social interaction for elderly healthcare: From Coaalas to AVICENA". In: *Journal of Medical Systems* Special Issue 2015: Agent-Empowered HealthCare Systems, pp. 1–20.

Governatori, Guido (2005). "Representing business contracts in RuleML". In: *International Journal of Cooperative Information Systems* 14.2-3, pp. 181–216. DOI: WorldScientificPublishingCompany. URL: http://espace.library.uq.edu.au/eserv.php?pid=UQ:9617&dsID=coala.pdf.

Governatori, Guido and Rotolo, Antonino (2010). "Norm compliance in business process modeling". In: *Semantic Web Rules*, pp. 194–209. URL: http://www.springerlink.com/index/R182X5732MX0661V.pdf.

Governatori, Guido; Hulstijn, Joris; Riveret, Régis and Rotolo, Antonino (2007). "Characterising deadlines in temporal modal defeasible logic". In: *AI 2007: Advances* *in Artificial Intelligence*, pp. 486–496. URL: http://www.springerlink.com/index/v6215747633972w0.pdf.

Grangard, Anders; Eisenberg, Brian; Nickull, Duane; Barham, Colin; Boseman, Al; Barret, Christian; Brooks, Dick; Casanave, Cory; Cunningham, Robert; Ferris, Christopher; Kacandes, Peter and Ketels, Kris (2001). "ebXML Technical Architecture Specification". In: *ebxml.org*. URL: http://www.ebxml.org/project_teams/technical_arch/private/index_files/ebXML_TA_v0.9.doc.

Grossi, Davide (2007). "Designing invisible handcuffs: Formal investigations in institutions and organizations for multi-agent systems". PhD thesis. Universiteit Utrecht. ISBN: 978-90-393-4619-8. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=13868003175764777892.

Grossi, Davide; Meyer, John-Jules Ch and Dignum, Frank (2006). "Counts-as: Classification or constitution? an answer using modal logic". In: *DEON 2006, LNAI* 4048, pp. 115–130. URL: http://www.springerlink.com/index/0306j05g58797w01.pdf.

— (2008). "The many faces of counts-as: A formal analysis of constitutive rules". In: *Journal of Applied Logic*. URL: http://linkinghub.elsevier.com/retrieve/pii/S1570868307000559.

Grossi, Davide; Aldewereld, Huib; Vázquez-Salceda, Javier and Dignum, Frank (2006). "Ontological aspects of the implementation of norms in agent-based electronic institutions". English. In: *Computational and Mathematical Organization Theory* 12.2-3, pp. 251–275. DOI: 10.1007/s10588-006-9546-6. URL: http://www.springerlink.com/index/10.1007/s10588-006-9546-6.

Groth, Paul; Luck, Michael and Moreau, Luc (2004). "Formalising a protocol for recording provenance in grids". In: *Proc. of the UK OST e-Science second All Hands Meeting*. URL: http://twiki.gridprovenance.org/pub/PASOA/PublicationStore/PReP-AHM04.pdf.

Hannoun, Mahdi; Boissier, Olivier; Sichman, Jaime S and Sayettat, Claudette (2000). "MOISE: An Organizational Model for Multi-agent Systems". English. In: *Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 156–165. ISBN: 978-3-540-41276-2. DOI: 10.1007/3-540-44399-1_17. URL: http://link.springer.com/10.1007/3-540-44399-1_17.

Hilpinen, Risto (1971). "Deontic Logic: Introductory and Systematic Readings". In: p. 184. URL: http://books.google.es/books?id=liDXAAAAMAAJ.

Hübner, Jomi F; Boissier, Olivier and Bordini, Rafael H (2009). "Normative programming for organisation management infrastructures". In: *Workshop on Coordination, Organization, Institutions and Norms in Agent Systems in Online Communities (COIN@MALLOW 2009)*.

Ibrhaim, Mamdouh; Holley, Kerrie; Josuttis, Nicolai M; Michelson, Brenda; Thomas, Dave and deVadoss, John (2007). "The future of SOA: what worked, what didn't, and where is it going from here?" In: *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion*. URL: http://portal.acm.org/citation.cfm?id=1297846.1297975.

Jakob, Michal; Pěchouček, Michal; Miles, Simon; Luck, Michael; Oren, Nir; Kollingbaum, Martin; Vázquez-Salceda, Javier; Storms, Patrick; Chabera, Jiri; Holt, Camden and Dehn, Martin (2008). "Case studies for contract-based systems". In: *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 55–62. URL: http://dl.acm.org/citation.cfm?id=1402795.1402806.

Jiang, Jie; Dignum, Virginia and Tan, Yao-Hua (2012). "An Agent-Based Inter-organizational Collaboration Framework: OperA+". English. In: *Coordination, Organizations, Institutions, and Norms in Agent System VII*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 58–74. ISBN: 978-3-642-35544-8. DOI: 10.1007/978-3-642-35545-5_4. URL: http://link.springer.com/10.1007/978-3-642-35545-5_4.

Jones, Andrew J I and Sergot, Marek J (1996). "A Formal Characterisation of Institutionalised Power". In: *Logic Journal of IGPL*. URL: http://jigpal.oxfordjournals.org/cgi/content/abstract/4/3/427.

Kanger, Stig and Stenlund, Sören (1974). "Logical Theory and Semantic Analysis: Essays Dedicated to Stig Kanger on His Fiftieth Birthday". In: *Springer, ISBN 9027704384*. URL: http://books.google.com/books?hl=en&lr=&ie=UTF-8&id=42IaYn5exX0C&oi=fnd&pg=PA1&dq=porn+logical+theory&ots=-bKG9erlqR&sig=5KfKuBopiKdG-Stmtkh3p7JHhwE.

Keller, Alexander and Ludwig, Heiko (2003). "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services". In: *Journal of Network and Systems Management*. URL: http://www.springerlink.com/index/M111304202683828.pdf.

Kenney, L Frank and Plummer, Daryl C (2008). "Magic Quadrant for Integrated SOA Governance Technology Sets, 2007". In: *Gartner RAS Core Research Note G00153858*, p. 16.

Kesselman, Carl and Foster, Ian (2004). "The Grid: Blueprint for a New Computing Infrastructure". In: *Morgan Kaufmann*. URL: http://books.google.com/books?hl=en&lr=&ie=UTF-8&id=8-0BofIhoU0C&oi=fnd&pg=PR7&dq=the+grid:+blueprint&ots=mrWkwB2VUp&sig=VULGDP3QNjw2AnWTcpzkpt6zOvc.

Kollingbaum, Martin (2005). "Norm-governed Practical Reasoning Agents". In: *Ph.D. Dissertation*.

Kolp, Manuel and Wautelet, Yves (2011). "A Social Framework for Software Architectural Design". In: *Handbook of Research on Socio-Technical Design and Social Networking Systems*. Ed. by Brian Whitworth, p. 1034. URL: https://books.google.co.uk/books/about/Handbook_of_Research_on_Socio_Technical.html?id=JsCPblyKEoIC.

Koo, Jarok (2008). "A Study on the Model Checking for Deontic Logic". In: *Convergence and Hybrid Information Technology, 2008. ICCIT '08. Third International Conference on*. IEEE, pp. 832–835. ISBN: 978-0-7695-3407-7. DOI: 10.1109/ICCIT.2008.240. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4682348&contentType=Conference+Publications&matchBoolean%3Dtrue%26rowsPerPage%

3D30%26searchField%3DSearch_All%26queryText%3D%28deontic+AND+%22model+checking%22%29.

Korsgaard, Christine M (2008). *The Constitution of Agency*. English. Essays on Practical Reason and Moral Psychology. Oxford University Press. ISBN: 0191564591. URL: http://books.google.es/books?id=wvR7KMLsY6kC&printsec=frontcover&dq=the+constitution+of+agency&hl=&cd=1&source=gbs_api.

Kozlenkov, Alexander and Schroeder, Michael (2004). "PROVA: Rule-Based Java-Scripting for a Bioinformatics Semantic Web". In: *Data Integration in the Life Sciences, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, ISBN 978-3-540-21300-0* 2994/2004, pp. 17–30. URL: http://www.springerlink.com/index/W7QCYMQBPV0UA181.pdf.

Kröger, Fred and Merz, Stephan (2008). "First-Order Linear Temporal Logic". English. In: *Temporal Logic and State Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 153–179. ISBN: 978-3-540-67401-6. DOI: 10.1007/978-3-540-68635-4_5. URL: http://link.springer.com/10.1007/978-3-540-68635-4_5.

Kyas, Marcel; Prisacariu, Cristian and Schneider, Gerardo (2008). "Run-time monitoring of electronic contracts". In: *Springer Berlin / Heidelberg* Proceedings of 6th International Symposium on Automated Technology for Verification and Analysis, Lecture Notes in Computer Science 5311, pp. 397–407. URL: http://www.springerlink.com/index/e5362773r8637tng.pdf.

Lam, Joey S C; Vasconcelos, Wamberto; Guerin, Frank; Corsar, David; Chorley, Alison H.; Norman, T J; Vázquez-Salceda, Javier; Panagiotidi, Sofia; Confalonieri, Roberto; Gómez-Sebastià, Ignasi; Hidalgo, Soraya; Alvarez-Napagao, Sergio; Nieves, Juan Carlos; Palau Roig, Manel; Ceccaroni, Luigi; Aldewereld, Huib; Dignum, Frank; Penserini, Luigi; Padget, Julian; Vos, Marina de; Andreou, D; Cliffe, Owen; Staikopoulos, Athanasios; Popescu, R; Clarke, S; Sergeant, P; Reed, C; Quillinan, Thomas and Nieuwenhuis, K (2009). "ALIVE: A Framework for Flexible and Adaptive Service Coordination". In: *Agents for Educational Games and Simulations*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 236–239. ISBN: 978-3-642-10202-8. DOI: 10.1007/978-3-642-10203-5_21. URL: http://link.springer.com/10.1007/978-3-642-10203-5_21.

Leite, João and Soares, Luís (2006). "Evolving characters in role playing games". In: *Cybernetics and Systems, Proceedings of the th European Meeting on Cybernetics and Systems Research EMCSR* 2, pp. 515–520. URL: http://www.researchgate.net/profile/Joao_Leite4/publication/228881113_Evolving_characters_in_role_playing_games/links/0deec52bcc5ab0fb7c000000.pdf.

Levinson, Stephen C (1983). *Pragmatics (Cambridge Textbooks in Linguistics)*. Cambridge University Press. DOI: 10.1234/12345678. URL: http://www.citeulike.org/group/2314/article/513907.

Lindahl, Lars (2001). "Stig Kanger's Theory of Rights". In: *Collected Papers of Stig Kanger with Essays on His Life and Work, Springer, ISBN: 1402001118*. URL: http://books.google.

com/books?hl=en&lr=&ie=UTF-8&id=U4w4BQy5-nIC&oi=fnd&pg=PA151&dq=porn+logical+theory&ots=LhpLS2KiAo&sig=mW_YkFOFxQTG8kA6_QDzkAjo1rs.

Linthicum, David (2008). "Defining SOA Governance". In: *InfoWorld*, p. 3.

Lokhorst, Gert-Jan C (1999). "Ernst Mally's Deontik (1926)". In: *Notre Dame Journal of Formal Logic* 40.2, pp. 273–282. URL: http://www.projecteuclid.org/Dienst/Repository/1.0/Disseminate/euclid.ndjfl/1038949542/body/pdfview.

Lomuscio, Alessio; Qu, Hongyang and Raimondi, Franco (2009). *MCMAS: A Model Checker for the Verification of Multi-Agent Systems*. Ed. by David Hutchison; Takeo Kanade; Josef Kittler; Jon M Kleinberg; Friedemann Mattern; John C Mitchell; Moni Naor; Oscar Nierstrasz; C Pandu Rangan; Bernhard Steffen; Madhu Sudan; Demetri Terzopoulos; Doug Tygar; Moshe Y Vardi; Gerhard Weikum; Ahmed Bouajjani and Oded Maler. Springer Berlin Heidelberg. Vol. 5643. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-02657-7. DOI: 10.1007/978-3-642-02658-4_55. URL: http://www.springerlink.com/index/10.1007/978-3-642-02658-4_55.

Long, Edmund (2007). "Enhanced NPC behaviour using goal oriented action planning". In: *PhD Thesis, University of Abertay-Dundee*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.8964&rep=rep1&type=pdf.

López, Fabiola López y and Luck, Michael (2002). "Towards a Model of the Dynamics of Normative Multi-Agent Systems". In: *Proceedings of the Int. Workshop on Regulated Agent-Based Social Systems*. URL: http://www.dcs.kcl.ac.uk/staff/mml/publications/assets/rasta02.pdf.

López, Fabiola López y; Luck, Michael and d'Inverno, Mark (2001). "A framework for norm-based inter-agent dependence". In: *Proceedings of The Third Mexican International Conference on Computer Science*, pp. 31–40. URL: http://citeseer.comp.nus.edu.sg/cache/papers/cs/23409/http:zSzzSzwww.ecs.soton.ac.ukzSz~mmlzSzpaperszSzenc01.pdf/lopez01framework.pdf.

— (2004). "Normative agent reasoning in dynamic societies". In: *Joint Conference on Autonomous Agents and ...* URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=16915294091999465116related:nE7kCFsyv-oJ.

Luck, Michael; McBurney, Peter; Shehory, Onn and Willmott, Steven (2005). *Agent Technology Roadmap: A roadmap for Agent Based Computing*. AgentLink Community. URL: http://scholar.google.com/scholar?q=related:wuskFMioXKEJ:scholar.google.com/&hl=en&num=20&as_sdt=0,5.

Ludwig, Heiko; Dan, Asit and Kearney, Robert (2004). "Cremona: an architecture and library for creation and monitoring of WS-agreements". In: *Proceedings of the 2nd international conference on Service oriented computing, New York, NY, USA, ISBN:1-58113-871-7*, pp. 65–74. URL: http://portal.acm.org/citation.cfm?id=1035178.

Ludwig, Heiko; Keller, Alexander; Dan, Asit; King, Richard P and Franck, Richard (2003). "Web Service Level Agreement (WSLA) Language Specification". In: *IBM Corporation*. URL: http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

MacKenzie, C Matthew; Laskey, Ken; McCabe, Francis; Brown, Peter F and Metz, Rebekah (2006). “Reference Model for Service Oriented Architecture 1.0”. In: *OASIS Committee Specification 1*, p. 31.

Manes, A T (2009). *Manes: SOA is dead; long live services*. Blogbeitrag [05.01. 2009]. URL: http://scholar.google.com/scholar?q=related:GF23I5ycGKIJ:scholar.google.com/&hl=en&num=20&as_sdt=0,5.

Matos, Noyda and Sierra, Carles (1999). “Evolutionary Computing and Negotiating Agents”. In: *AMET-98, LNAI 1571, Lectures Notes on Computer Science, Springer-Verlag Berlin Heidelberg*, pp. 126–150. URL: http://www.springerlink.com/index/28CWJV6YKCEMJB2L.pdf.

Matyska, Ludek; Krenek, Ales; Ruda, Miroslav; Sitera, Jiri; Kouril, Daniel; Vocu, Michal; Pospisil, Jan; Mulac, Milos and Salvet, Zdenek (2007). “Job Tracking on a Grid—the Logging and Bookkeeping and Job Provenance Services”. In: *CESNET technical report number 9/2007*. URL: http://www.cesnet.cz/doc/techzpravy/2007/grid-job-tracking/grid-job-tracking.ps.gz.

McCarthy, Dennis; Dayal, Umeshwar; McCarthy, Dennis and Dayal, Umeshwar (1989). *The architecture of an active database management system*. English. Vol. 18. ACM. ISBN: 0-89791-317-5. DOI: 10.1145/66926.66946. URL: http://portal.acm.org/citation.cfm?doid=66926.66946.

McNamara, Paul and Prakken, Henry (1999). “Norms, Logics and Information Systems: New Studies in Deontic Logic and Computer Science”. In: *IOS Press*. URL: http://books.google.com/books?hl=en&lr=&ie=UTF-8&id=Efz2tm2BwkIC&oi=fnd&pg=PR7&dq=P.+McNamara+and+H.+Prakken&ots=xp9XGh5t3g&sig=oXM_eFURiAu5hjdKPCnECNzEAOw.

Meneguzzi, Felipe and Luck, Michael (2009). *Norm-based behaviour modification in BDI agents*. AAMAS ’09. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. ISBN: 978-0-9817381-6-1. URL: http://dl.acm.org/citation.cfm?id=1558013.1558037.

Meneguzzi, Felipe; Rodrigues, Odinaldo; Oren, Nir; Vasconcelos, Wamberto and Luck, Michael (2015). “BDI reasoning with normative considerations”. English. In: *Engineering Applications of Artificial Intelligence* 43, pp. 127–146. DOI: 10.1016/j.engappai.2015.04.011. URL: http://linkinghub.elsevier.com/retrieve/pii/S0952197615000925.

Menzel, Michael; Thomas, Ivonne; Wolter, Christian and Meinel, Christoph (2007). “SOA Security-Secure Cross-Organizational Service Composition”. In: *Proceedings of the Stuttgarter Softwaretechnik Forum (SSF), Fraunhofer IRB-Verlag, Stuttgart, Germany, ISBN: 978-3-8167-7493-8*, pp. 41–53. URL: http://www.hpi.uni-potsdam.de/fileadmin/hpi/FG_ITS/papers/Menzel_SSF2007.pdf.

Meyer, John-Jules Ch and Wieringa, R (1991). “Deontic Logic in Computer Science: Normative System Specification”. In: *Procs. of MAAMAW’01*. URL: http://www.di.unito.it/~guido/violator/2004-old/Lpar/yourwish.bib.gz.

Michlmayr, Anton; Rosenberg, Florian; Platzer, Christian; Treiber, Martin and Dustdar, Schahram (2007). “Towards recovering the broken SOA triangle: a software engineering perspective". In: *2nd international workshop on Service oriented software engineering: in conjunction with the 6th ESEC/FSE joint meeting*. New York, New York, USA: ACM, pp. 22–28. ISBN: 978-1-59593-723-0. DOI: 10.1145/1294928.1294934. URL: http://portal.acm.org/citation.cfm?doid=1294928.1294934.

Millington, Ian and Funge, John (2009). "Artificial Intelligence for Games". In: *Morgan Kaufmann*. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=10499474543559257567related:3y0bAXigtZEJ.

Modgil, Sanjay; Oren, Nir; Faci, Noura; Meneguzzi, Felipe; Miles, Simon and Luck, Michael (2015). "Monitoring compliance with E-contracts and norms". English. In: *Artificial Intelligence and Law* 23.2, pp. 161–196. DOI: 10.1007/s10506-015-9167-9. URL: http://link.springer.com/article/10.1007/s10506-015-9167-9/fulltext.html.

Monks, Robert A G and Minow, Nell (2004). "Corporate Governance". In: *Blackwell Publishing*. URL: http://books.google.com/books?hl=en&lr=&ie=UTF-8&id=RGmHAVPhmRwC&oi=fnd&pg=PR15&dq=corporate+governance+definition&ots=h6zYqP8f5D&sig=KuLgwbezObOjDfq2etZQXgha7JY.

Moreau, Luc; Freire, Juliana; Futrelle, Joe; McGrath, Robert E; Myers, Jim and Paulson, Patrick (2008). "The Open Provenance Model: An Overview". English. In: *Provenance and Annotation of Data and Processes*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 323–326. ISBN: 978-3-540-89964-8. DOI: 10.1007/978-3-540-89965-5_31. URL: http://link.springer.com/10.1007/978-3-540-89965-5_31.

Nareyek, Alexander (2007). "Game AI Is Dead. Long Live Game AI!" English. In: *Intelligent Systems* 66.4, pp. 1338–1343. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=6804820867560721766related:Zk0fi4aUb14J.

Nau, Dana; Au, Tsz-Chiu; Ilghami, Okhtay; Kuter, Ugur; Murdock, J William; Wu, Dan and Yaman, Fusun (2003). "SHOP2: An HTN planning system". In: *Journal of Artificial Intelligence Research* 20, pp. 379–404. URL: https://www.aaai.org/Papers/JAIR/Vol20/JAIR-2013.pdf.

Newell, Allen; Rosenbloom, Paul S. and Laird, John E (1987). "SOAR: An architecture for general intelligence". English. In: *Carnegie-Mellon University Technical Report* 66.4, pp. 1281–1286. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=15352123283218065173related:FWc3fMmyDdUJ.

Nezhad, Hamid R Motahari; Benatallah, Boualem; Casati, Fabio and Toumani, Farouk (2006). "Web Services Interoperability Specifications". In: *Computer, IEEE Computer Society, Los Alamitos, CA, USA, ISSN: 0018-9162* 39.5, pp. 24–32. URL: http://doi.ieeecomputersociety.org/10.1109/MC.2006.181.

Nilsson, Nils J and Fikes, Richard E (1970). "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Storming Media LLC*. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=18338894177823518748related:HHCmUhfXgP4J.

Noriega, Pablo (1997). “Agent-Mediated Auctions: The Fishmarket Metaphor”. In: *IIIA Phd Monography* 8.

North, Douglass C; Nørgaard, Asbjørn Sonne and Swedberg, Richard (1990). “Institutions, Institutional Change and Economic Performance”. In: *Cambridge University Press*. URL: http://books.google.com/books?hl=en&lr=&ie=UTF-8&id=oFnWbTqgNPYC&oi=fnd&pg=PR6&dq=Institutions,+Instutional+Change+and+Econonomic+Performance&ots=sWkpP8JqO5&sig=3tXS3C7OvyV9Hmf7U2KDNGBcqKw.

Oliveira, Eunice; Cardoso, Henrique; Urbano, Joana and Rocha, Ana Paula (2014). *Trustworthy agents for B2B operations under Normative environment*. English. IEEE. ISBN: 978-1-4799-5457-5. DOI: 10.1109/ICSAI.2014.7009295. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7009295.

Oren, Nir; Panagiotidi, Sofia; Vázquez-Salceda, Javier; Modgil, Sanjay; Luck, Michael and Miles, Simon (2009). “Towards a formalisation of electronic contracting environments”. In: LNAI 5428, pp. 156–171. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=2600664954751297477related:xSfCV_9qFyQJ.

Orkin, Jeff (2006). “Three states and a plan: the AI of FEAR”. In: *Proc. of the 2006 Game Developers Conference*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.8551&rep=rep1&type=pdf.

Ostrom, Elinor (1986). “An agenda for the study of institutions”. In: *Public Choice*. URL: http://www.springerlink.com/index/T67314712U831845.pdf.

Panagiotidi, Sofia and Vázquez-Salceda, Javier (2011). “Norm-Aware Planning: Semantics and Implementation”. In: *2011 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*. IEEE, pp. 33–36. ISBN: 978-1-4577-1373-6. DOI: 10.1109/WI-IAT.2011.249. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6040698.

Panagiotidi, Sofia; Vázquez-Salceda, Javier; Alvarez-Napagao, Sergio; Ortega-Martorell, Sandra; Willmott, Steven; Confalonieri, Roberto and Storms, Patrick (2008). “Intelligent Contracting Agents Language”. In: Volume 4: Proceedings of the AISB 2008 Symposium on Behaviour Regulation in Multi-agent Systems.Aberdeen, Scotland, pp. 49–55. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=related:GdxvHmx7_OUJ:scholar.google.com/.

Paschke, Adrian (2005). “RBSLA A declarative Rule-based Service Level Agreement Language based on RuleML”. In: *Computational Intelligence for Modelling*. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1631486.

Paschke, Adrian; Dietrich, Jens and Kuhla, Karsten (2005). “A Logic Based SLA Management Framework”. In: *Proceedings of the 4th Semantic Web Conference (ISWC 2005)* Galway, Ireland, pp. 68–83. URL: http://ibis.in.tum.de/staff/paschke/docs/ISWC05_Paschke_final.pdf.

Paurobally, Shamimabi; Tamma, Valentina and Wooldridge, Michael (2005). “Cooperation and Agreement between Semantic Web Services”. In: *W3C Workshop on Frame-*

works for Semantics in Web Services. URL: http://www.csc.liv.ac.uk/~valli/Papers/swsv2.pdf.

Pautasso, Cesare; Zimmermann, Olaf and Leymann, Frank (2008). "Restful web services vs. big' web services: making the right architectural decision". In: *Proceeding of the 17th international conference*. New York, New York, USA: ACM, pp. 805–814. ISBN: 978-1-60558-085-2. DOI: 10.1145/1367497.1367606. URL: http://portal.acm.org/citation.cfm?doid=1367497.1367606.

Pörn, Ingmar (1974). "Some Basic Concepts of Action". In: *Logical Theory and Semantic Analysis: Essays Dedicated to Stig Kanger on His Fiftieth Birthday, Springer, ISBN: 9027704384*. URL: http://books.google.com/books?hl=en&lr=&ie=UTF-8&id=42IaYn5exX0C&oi=fnd&pg=PT107&dq=porn+logical+theory&ots=-bKG9erlqR&sig=OQAS9sKP4Gt4rzFAuvktrY_c1Kk.

Posner, Richard A (2009). *The Problematics of Moral and Legal Theory*. Harvard University Press. URL: https://books.google.co.uk/books/about/The_Problematics_of_Moral_and_Legal_Theo.html?id=E4Oyzn1oWtsC.

Prakken, Henry and Sergot, Marek (1996). "Contrary-to-duty obligations". English. In: *Studia Logica* 57.1, pp. 91–115. DOI: 10.1007/BF00370671. URL: http://link.springer.com/10.1007/BF00370671.

— (1997). "Dyadic Deontic Logic and Contrary-to-Duty Obligations". English. In: *Defeasible Deontic Logic*. Dordrecht: Springer Netherlands, pp. 223–262. ISBN: 978-94-015-8851-5. DOI: 10.1007/978-94-015-8851-5_10. URL: http://www.springerlink.com/index/10.1007/978-94-015-8851-5_10.

Prisacariu, Cristian and Schneider, Gerardo (2009). "Abstract specification of legal contracts". In: *ICAIL '09: Proceedings of the 12th International Conference on Artificial Intelligence and Law*. New York, New York, USA: ACM Request Permissions, pp. 218–219. ISBN: 9781605585970. DOI: 10.1145/1568234.1568262. URL: http://portal.acm.org/citation.cfm?id=1568234.1568262&coll=DL&dl=ACM&CFID=93104618&CFTOKEN=96925591.

Rao, Anand S and Georgeff, Michael P (1995). "BDI Agents: From Theory to Practice". In: *Proceedings of the First International Conference on Multiagent Systems*, pp. 312–319. URL: http://www.aaai.org/Papers/ICMAS/1995/ICMAS95-042.pdf.

Richard Scott, W (1998). "Organizations: Rational, Natural, and Open Systems". In: *Prentice Hall*, p. 416. URL: http://books.google.com/books?id=8XfGHQAACAAJ&printsec=frontcover.

— (2001). "Institutions and Organizations". In: *Foundations for Organizational Science*, p. 255. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=6165349868156549022related:nvd4Jwq5j1UJ.

Riemsdijk, M Birna van; Dennis, Louise; Fisher, Michael and Hindriks, Koen V (2015). *A Semantic Framework for Socially Adaptive Agents: Towards strong norm compliance*. International Foundation for Autonomous Agents and Multiagent Systems. ISBN: 978-1-4503-3413-6. URL: http://dl.acm.org/citation.cfm?id=2772879.2772935.

Rodríguez-Aguilar, Juan Antonio (2003). “On the Design and Construction of Agent-mediated Institutions”. In: *PhD Thesis*. URL: http://www.tdr.cesca.es/TDX-1202103-150907/index_cs.html.

Rosenau, James N and Czempiel, Ernst Otto (1992). “Governance Without Government: Order and Change in World Politics”. In: *Cambridge University Press*. URL: http://books.google.com/books?hl=en&lr=&ie=UTF-8&id=lexwc-iBDs4C&oi=fnd&pg=PR9&dq=governance&ots=wLlGCUY8ay&sig=5RuVo2GBhPM-ypc8Ykk3r0kzxfo.

Santos, Filipe and Carmo, José (1996). “Indirect Action, Influence and Responsibility”. In: *In Deontic Logic, Agency and Normative Systems, M. Brown and J. Carmo (eds), Springer*, pp. 194–215. URL: http://iscte.pt/~fas/deon96.pdf.

Santos, Filipe A A; Jones, Andrew J I and Carmo, José (1997). “Action concepts for describing organised interaction”. In: *System Sciences*. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=663195.

Schmidt, Marc-Thomas; Hutchison, Beth; Lambros, Peter and Phippen, Rob (2005). “The Enterprise Service Bus: Making service-oriented architecture real”. In: *IBM Systems Journal*. URL: https://www.research.ibm.com/journal/sj/444/schmidt.html.

Searle, John R (1969). “Speech acts: An essay in the philosophy of language”. In: *books.google.com*. URL: http://books.google.com/books?hl=en&lr=&id=t3_WhfknvF0C&oi=fnd&pg=PA3&dq=searle+speech+acts&ots=0QiW9NO9T2&sig=NlcNaRbkblfFIQbY3RJVpXEX-z0.

— (1985). *Expression and Meaning*. Studies in the Theory of Speech Acts. Cambridge University Press. URL: https://books.google.co.uk/books/about/Expression_and_Meaning.html?id=dhf27-nv7pkC.

— (2005). “What is an institution?” In: *Journal of Institutional Economics* 1.1, pp. 1–22. DOI: 10.1017/S1744137405000020. URL: http://scholar.google.com/scholar?q=related:88MB1bBwPioJ:scholar.google.com/&hl=en&num=30&as_sdt=0,5.

— (2009). *Making the Social World: The Structure of Human Civilization*. Oxford University Press. URL: https://books.google.co.uk/books/about/Making_the_Social_World_The_Structure_of.html?id=kz6R0eDZ5OEC.

Sergot, Marek (2010). “(C+)++ : An action language for modelling norms and institutions”. In: pp. 1–88.

Simonsson, Mârten and Johnson, Pontus (2006). “Defining IT Governance-A Consolidation of Literature”. In: *Proceedings of the 18th Conference on Advanced Information Systems Engineering Engineering*. URL: http://www.ics.kth.se/Publikationer/Working%20Papers/EARP-WP-2005-MS-04.pdf.

Simpson, John and Weiner, Edmund (2003). “Oxford English Dictionary”. In: *Oxford University Press*. URL: http://www.oup.co.uk/highlights/bestsellers/.

Sommerville, I (2006). “Software Engineering”. In: *Pearson Education, ISBN: 0321313798*. URL: http://books.google.com/books?hl=en&lr=&ie=UTF-8&id=B7idKfL0H64C&oi=fnd&pg=PR5&dq=software+monitoring&ots=X1tVZIUbFP&sig=GYSRecEOrEmES65lJ-JMvkf4eDs.

Spivey, J M (1989). “The Z notation”. In: *Prentice Hall International (UK)*. URL: http://www.rose-hulman.edu/class/se/csse373/current/Resources/zrm.pdf.

Spronck, P; Ponsen, M and Sprinkhuizen-Kuyper, I (2006). “Adaptive game AI with dynamic scripting”. English. In: *Machine Learning* 66.4, pp. 1338–1343. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=17315633243202551546related:-n4ISrF8TfAJ.

Strano, Massimo; Molina-Jimenez, Carlos and Shrivastava, Santosh (2008). “A rule-based notation to specify executable electronic contracts”. In: *Springer Berlin / Heidelberg* Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web (RuleML2008), Lecture Notes in Computer Science 5321, pp. 81–88. URL: http://www.springerlink.com/index/f27454xuw6665678.pdf.

Tauriainen, Heikki (2006). “Automata and Linear Temporal Logic: Translations with Transition-based Acceptance”. PhD thesis. Helsinki University of Technology.

Tinnemeier, Nick; Dastani, Mehdi and Meyer, John-Jules Ch (2009). “Roles and norms for programming agent organizations”. In: *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)* 1.Budapest, Hungary, pp. 121–128.

Torre, L van der; Hulstijn, J; Dastani, Mehdi and Broersen, Jan (2004). “Specifying Multiagent Organizations”. In: *Deontic Logic in Computer Science*. URL: http://www.springerlink.com/index/v5fmwkyc9bufh6ec.pdf.

Trites, Gerald (2004). “Director responsibility for IT governance”. In: *International Journal of Accounting Information Systems*. URL: http://linkinghub.elsevier.com/retrieve/pii/S1467089504000089.

Tuomela, Raimo (1996). “Philosophy and distributed artificial intelligence: the case of joint intention”. In: *John Wiley Sixth-Generation Computer Technology Series*. URL: http://portal.acm.org/citation.cfm?id=239297.239334.

Vázquez-Salceda, Javier (2003). “The role of Norms and Electronic Institutions in Multi-Agent Systems applied to complex domains. The HARMONIA framework”. In: p. 242.

— (2004). “The Role of Norms and Electronic Institutions in Multi-Agent Systems: The HARMONIA Framework.” In: *PhD Thesis, Universitat Politècnica de Catalunya*.

Vázquez-Salceda, Javier; Aldewereld, Huib and Dignum, Frank (2004). “Implementing Norms in Multiagent Systems”. In: *G. Lindemann et al. (Eds.): MATES 2004, LNAI 3187, Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg*, pp. 313–327. URL: http://www.springerlink.com/index/06WJEY8X72V91H57.pdf.

Vázquez-Salceda, Javier and Alvarez-Napagao, Sergio (2009). “Using SOA Provenance to Implement Norm Enforcement in e-Institutions”. English. In: *Coordination, Organizations, Institutions and Norms in Agent Systems IV*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 188–203. ISBN: 978-3-642-00442-1. DOI: 10.1007/978-3-642-00443-8_13. URL: http://www.springerlink.com/index/j37549365404x865.pdf.

Vázquez-Salceda, Javier; Cortés, Ulises; Padget, Julian; López-Navidad, Antonio and Caballero, F (2003). “The organ allocation process: a natural extension of the Carrel Agent-Mediated Electronic Institution”. In: *AI Communications*. URL: http://iospress.metapress.com/index/JG0NV8L96L0RRLEQ.pdf.

Walter, Robert (1996). “Jörgensen’s dilemma and how to face it”. English. In: *Ratio Juris* 9.2, pp. 168–171. DOI: 10.1111/j.1467-9337.1996.tb00234.x. URL: http://onlinelibrary.wiley.com/doi/10.1111/j.1467-9337.1996.tb00234.x/abstract.

Webb, Phyl; Pollard, Carol and Ridley, Gail (2006). “Attempting to Define IT Governance: Wisdom or Folly?” In: *Hawaii International Conference on System Sciences*. URL: http://doi.ieeecomputersociety.org/10.1109/HICSS.2006.68.

webMethods (2006). “SOA Governance: Enabling Sustainable Success with SOA”. In: *webMethods, Inc.* URL: http://www1.webmethods.com/PDF/whitepapers/SOA_Governance.pdf.

Weill, Peter (2004). “Don’t Just Lead, Govern: How Top-Performing Firms Govern IT”. In: *MIS Quarterly Executive*. URL: http://web.mit.edu/cisr/working%20papers/cisrwp341.pdf.

Whitworth, Brian (2009). “The Social Requirements of Technical Systems”. In: *Handbook of Research on Socio-Technical Design and Social Networking Systems*. Ed. by Brian Whitworth. IGI Global. URL: https://books.google.co.uk/books/about/Handbook_of_Research_on_Socio_Technical.html?id=JsCPblyKEoIC.

Winikoff, Michael (2005). “Jack™ Intelligent Agents: An Industrial Strength Platform”. English. In: *Multi-Agent Programming*. Boston, MA: Springer US, pp. 175–193. ISBN: 978-0-387-24568-3. DOI: 10.1007/0-387-26350-0_7. URL: http://link.springer.com/10.1007/0-387-26350-0_7.

Wooldridge, Michael and Jennings, Nicholas R (1995). “Intelligent Agents: Theory and Practice”. English. In: *Knowledge engineering review* 66.4, pp. 1424–1431. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=7316328098967361322related:Kpcod6TRiGUJ.

Wright, G. H. von (1951). “Deontic logic”. In: *Mind, New Series* 60.237, pp. 1–15. DOI: 10.2307/2251395. URL: http://www.jstor.org/stable/2251395.

Xiao, Ding and Zhong, Xiaoan (2010). “Improving Rete algorithm to enhance performance of rule engine systems”. English. In: *2010 International Conference on Computer Design and Applications (ICCDA 2010)* 3, pp. V3–572–V3–575. DOI: 10.1109/ICCDA.2010.5541368. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5541368.

Zimmermann, Michael (2009). “OWL2Java”. In: URL: http://www.incunabulum.de/projects/it/owl2java.

Selected publications

Journal Papers

Gómez-Sebastià, Ignasi; Garcia-Gasulla, Dario and Alvarez-Napagao, Sergio (2011). "Society of situated agents for adaptable eldercare". In: *ERCIM News* 87, pp. 23–24. URL: http://ercim-news.ercim.eu/en87/special/a-society-of-situated-agents-for-adaptable-eldercare.

This article describes a use case in which normative and organisational structures are used to implement an adaptive platform for eldercare management integrating humans, sensors and actuators. I participated in the definition of the conceptual framework and in the modelling of the normative structures. [This work is a consequence and application of the work presented in Chapters 5, 6 and 7, and will be used as a use case in Chapter 9 in the final document.]

Gómez-Sebastià, Ignasi; Moreno, Jonathan; Alvarez-Napagao, Sergio; Garcia-Gasulla, Dario; Barrué, Cristian and Cortés, Ulises (2015). "Situated agents and humans in social interaction for elderly healthcare: From Coaalas to AVICENA". In: *Journal of Medical Systems* Special Issue 2015: Agent-Empowered HealthCare Systems, pp. 1–20.

This paper presents a proof of concept describing a social network for materialising complex relationships in eldercare contexts. This work partly builds upon previous work done in normative structures for eldercare management, and therefore my contribution is the participation on the design and modelling of the normative structures underlying the social network. [This work is a consequence and application of the work presented in Chapters 5, 6 and 7, and is used as a use case in Chapter 9.]

Kifor, Tamás; Varga, Laszlo Zs; Vázquez-Salceda, Javier; Alvarez-Napagao, Sergio; Willmott, Steven; Miles, Simon and Moreau, Luc (2006). "Provenance in Agent-Mediated Healthcare Systems". In: *IEEE Intelligent Systems* 21.6, pp. 38–46. DOI: 10.1109/MIS.2006.119. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4042534.

This paper explains in detail how provenance mechanisms were applied in the context of organ transplant management in order to automatically verify the compliance of regulations and protocols, while preserving the privacy of the parties involved. In this work, I designed and implemented several components used for the use case: the agent platform along with the com-

munication and the event models. Also, I was responsible for translating the regulations and protocols to rules to be fed to the compliance verification rule engine. [This work is used as a part of the motivation in Section 4.1, and is the basis of the architecture presented in Section 8.2.]

Book Chapters

Vázquez-Salceda, Javier; Alvarez-Napagao, Sergio; Kifor, Tamás; Varga, Laszlo Zs; Miles, Simon; Moreau, Luc and Willmott, Steven (2007). "EU PROVENANCE Project: An Open Provenance Architecture for Distributed Applications". In: *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*, pp. 55–64.

This book chapter describes Provenance as a valid mechanism for causal-historical documentation and regulation verification of complex distributed processes, presenting a use case based on organ transplant management as a use case. My contribution was in the transformation of real-world concepts into agents in an agent platform, and in the translation of relevant events, regulations and protocols into provenance concepts and rules that were key in proving the concept. [This work is used as a part of the motivation in Section 4.1, and is the basis of the architecture presented in Section 8.2.]

Conference and Workshop Publications Related to the PhD Thesis

Aldewereld, Huib; Alvarez-Napagao, Sergio; Dignum, Frank and Vázquez-Salceda, Javier (2009). "Engineering Social Reality with Inheritance Relations". English. In: *Lecture Notes in Computer Science*. Ed. by David Hutchison; Takeo Kanade; Josef Kittler; Jon M Kleinberg; Friedemann Mattern; John C Mitchell; Moni Naor; Oscar Nierstrasz; C Pandu Rangan; Bernhard Steffen; Madhu Sudan; Demetri Terzopoulos; Doug Tygar; Moshe Y Vardi; Gerhard Weikum; Alessandro Acquisti; Sean W Smith and Ahmad-Reza Sadeghi. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 116–131. isbn: 978-3-642-10202-8. doi: `10.1007/978-3-642-10203-5_11`. url: `http://link.springer.com/10.1007/978-3-642-10203-5_11`.

This paper presents a study on the several aspects of counts-as rules and how they can be translated into valid formalisms by the use of inheritance relations. My contribution consisted in the grounding of inheritance relations into logically-sound and efficient production systems, the implementation of the use case, as well as in participating in the formalisation of inheritance relations. [This work is the basis of Sections 6.1, 6.2 and 7.3.]

— (2010). "Making norms concrete". In: *AAMAS '10: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems* Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, pp. 807–814. URL: http://dl.acm.org/citation.cfm?id=1838206.1838314.

This paper builds upon our own previous research on the various aspects of counts-as rules to propose a formalism for the materialisation of norms using inheritance relations and the grounding of deontics on counts-as rules. My contribution was the formalisation of such grounding on production systems' semantics, providing a logically-sound design and an example implementation operationalising such semantics along with a use case. [This work is the basis of Sections 6.1, 6.2 and 7.3.]

Alvarez-Napagao, Sergio and Vázquez-Salceda, Javier (2008). "Using Provenance to implement a Norm Enforcement Mechanism for Agent-Mediated Healthcare Systems". In: *Proceedings of the Fifth Workshop on Agents Applied in Health Care at AAMAS'08, Estoril, Portugal*, p. 8.

This paper presents the results of the design and implementation of a norm-enforcement architecture based on the transformation of events, regulations and protocols into provenance assertions and production systems. My contribution was the design of the architecture and the implementation of all the components, including the formal translations from concepts to assertions and rules. [This work is used as a part of the motivation in Section 4.1, and is the basis of the architecture presented in Section 8.2.]

Alvarez-Napagao, Sergio; Vázquez-Salceda, Javier; Kifor, Tamás; Varga, Laszlo Zs and Willmott, Steven (2006). "Applying provenance in distributed organ transplant management". In: *International Provenance and Annotation workshop (IPAW 2006), 3-5 May 2006, Chicago, USA, ISBN 978-3-540-46302-3*. URL: http://twiki.grimoires.org/pub/Provenance/ProjectPublications/IPAW-OTM-EHCR.pdf.

This paper explains in detail how provenance mechanisms were applied in the context of organ transplant management in order to automatically verify the compliance of regulations and protocols, while preserving the privacy of the parties involved. In this work, I designed and implemented several components used for the use case: the agent platform along with the communication and the event models. Also, I was responsible for translating the regulations and protocols to rules to be fed to the compliance verification rule engine. [This work is used as a part of the motivation in Section 4.1, and is the basis of the architecture presented in Section 8.2.]

Alvarez-Napagao, Sergio; Cliffe, Owen; Vázquez-Salceda, Javier and Padget, Julian (2009). "Norms, organisations and semantic web services: The ALIVE approach". English. In: Coordination, Organization, Institutions and Norms in Agent Systems & On-line Communities (COIN@MALLOW'009), Proceedings of the Second Multi-Agent Logics, Languages, and Organisations Federated Workshops, Volume 494, pp. 1–2. ISSN: 1613-0073. URL: http://upcommons.upc.edu//handle/2117/14283.

This paper summarises how the ALIVE approach for the management of distributed systems fits the Service-Orientation model. My contribution in this work was the participation on the normative and the organisational part of the meta-model specification, along with the implementation of the event-management and monitoring. components. [This work describes some of the contributions presented in Section 5.2.]

Alvarez-Napagao, Sergio; Koch, Fernando; Gómez-Sebastià, Ignasi and Vázquez-Salceda, Javier (2011). “Making Games ALIVE: An Organisational Approach”. English. In: *Agents for Games and Simulations II*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 179–191. ISBN: 978-3-642-18180-1. DOI: 10.1007/978-3-642-18181-8_13. URL: http://link.springer.com/10.1007/978-3-642-18181-8_13.

This paper presents a proposal for the use of organisational structures to model concepts common to three types of fun games, in order to provide social concepts and a higher level of adaptivity in video-games. My contribution was the conceptual framework of the work, the state of the art, the architecture of the proposal and the implementation of the proofs of concept. [This work is an application of the contributions presented in Chapters 5, 6 and 7 and is the basis of the use case in Chapter 9.]

Alvarez-Napagao, Sergio; Gómez-Sebastià, Ignasi; Panagiotidi, Sofia; Tejeda-Gómez, Arturo; Oliva-Felipe, Luis and Vázquez-Salceda, Javier (2012). “Socially-Aware Emergent Narrative”. English. In: *Lecture Notes in Computer Science*. Ed. by Martin Beer; Cyril Brom; Frank Dignum and Von-Wun Soo. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 139–150. ISBN: 978-3-642-32325-6. DOI: 10.1007/978-3-642-32326-3_9. URL: http://www.lsi.upc.edu/~igomez/Papers/AEGS_2011.pdf.

This paper presents a framework for integrating social and normative structures into gaming emergent narrative engines, including a use case with organisational models and an architecture. My contribution was the state of the art, the conceptual framework of the work, and the design and implementation of the architecture. [This work is an application of the contributions presented in Chapters 5, 6 and 7 and is directly related to the use case in Chapter 9.]

Confalonieri, Roberto; Alvarez-Napagao, Sergio; Panagiotidi, Sofia; Vázquez-Salceda, Javier and Willmott, Steven (2008). “A Middleware Architecture for Building Contract-Aware Agent-Based Services”. English. In: *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–14. ISBN: 978-3-540-79967-2. DOI: 10.1007/978-3-540-79968-9_1. URL: http://www.springerlink.com/index/m84n55h118j86178.pdf.

This paper presents a middleware for integrating high-level specifications of contracts in Service-Oriented Architectures, by the use of underlying intelligent contract-aware agents. My contribution in this work consisted in the participation in the definition of the contracting language and the ontological layers, as well as the creation of a few contract-based interaction protocols. [This work builds upon the contributions presented in Section 5.1.]

Gómez-Sebastià, Ignasi and Alvarez-Napagao, Sergio (2012). "Towards Runtime Support for Norm Change from a Monitoring Perspective". In: Proceedings of the First International Conference on Agreement Technologies, Volume 918. URL: http://vi.ikt.ui.sav.sk/@api/deki/files/2119/=example.pdf.

This paper analyses the different aspects of norm dynamics and proposes a formal framework for dealing with each aspect at run-time. My contribution was the participation in the state of the art, the formalisation of the static aspect of norms, and also the participation in the conceptual formalisation of the operationalisation of the dynamic aspects. [This work builds upon the contributions presented in Chapters 5 and 6.]

Gómez-Sebastià, Ignasi; Alvarez-Napagao, Sergio and Vázquez-Salceda, Javier (2011). "A Distributed Norm Compliance Model". In: *Frontiers in Artificial Intelligence and Applications* Volume 232: Artificial Intelligence Research and Development, pp. 110–119. DOI: 10.3233/978-1-60750-842-7-110. URL: http://ebooks.iospress.nl/publication/6573.

This paper presents an algorithm for the distribution of norm monitoring tasks, based on the relationships between norms between themselves and norms and events and grounded on Strongly Connected Components. My main contribution to this work was the formalism for norms, its operational semantics. Also, the definition of norm dependency was a product of joint work with the other authors, especially contributing on the formalisation tasks. [This work builds upon the contributions presented in Chapters 5 and 6.]

Gómez-Sebastià, Ignasi; Alvarez-Napagao, Sergio; Garcia-Gasulla, Dario and Cortés, Ulises (2013). "Situated agents and humans in social interaction for elderly healthcare: the case of COAALAS ". In: *14th Conference on Artificial Intelligence in Medicine* Proceedings of the VIII Workshop on Agents Applied in Health Care (A2HC2013), pp. 105–119. URL: http://upcommons.upc.edu/e-prints/bitstream/2117/23299/1/Situated%20agents%20and%20humans%20in%20social%20interaction%20for%20elderly%20healthcare:The%20case%20of%20COAALAS.pdf.

This paper presents a proposal for an organisational framework for eldercare contexts involving patients, doctors, pharmacy providers along with sensors and actuators, with the objective of providing an adaptive fail-safe mechanism for improving eldercare management. My involvement in this work focused on the definition of the conceptual framework and the creation of the social, organisational and normative models, and on the grounding to the ALIVE meta-model. [This work is a consequence and application of the work presented in Chapters 5, 6 and 7, and is used as a use case in Chapter 9.]

Kifor, Tamás; Varga, Laszlo Zs; Alvarez-Napagao, Sergio; Vázquez-Salceda, Javier and Willmott, Steven (2006). "Privacy Issues of Provenance in Electronic Healthcare Record Systems". In: *First International Workshop on Privacy and Security in Agent-based Collaborative Environments (PSACE2006), Hakodate, Japan*. URL: http://twiki.

gridprovenance.org/pub/Provenance/ProjectPublications/EHCR-Prov-Privacy.pdf.

This paper enumerates the privacy issues in documenting the provenance of events in the context of healthcare processes, and the possible solutions for such issues. My participation in this work was focused on implementing the underlying agent platform of the organ transplant management use case used as an artifact in the paper analysis, along with the design and implementation of the translation of events, regulations and protocols to provenance assertions and productions systems. [This work is used as a part of the motivation in Section 4.1, and is the basis of the architecture presented in Section 8.2.]

Lam, Joey S C; Vasconcelos, Wamberto; Guerin, Frank; Corsar, David; Chorley, Alison H.; Norman, T J; Vázquez-Salceda, Javier; Panagiotidi, Sofia; Confalonieri, Roberto; Gómez-Sebastià, Ignasi; Hidalgo, Soraya; Alvarez-Napagao, Sergio; Nieves, Juan Carlos; Palau Roig, Manel; Ceccaroni, Luigi; Aldewereld, Huib; Dignum, Frank; Penserini, Luigi; Padget, Julian; Vos, Marina de; Andreou, D; Cliffe, Owen; Staikopoulos, Athanasios; Popescu, R; Clarke, S; Sergeant, P; Reed, C; Quillinan, Thomas and Nieuwenhuis, K (2009). "ALIVE: A Framework for Flexible and Adaptive Service Coordination". In: *Agents for Educational Games and Simulations*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 236–239. ISBN: 978-3-642-10202-8. DOI: 10.1007/978-3-642-10203-5_21. URL: http://link.springer.com/10.1007/978-3-642-10203-5_21.

This article reports the main advancements made during the ALIVE project. My contribution in this work was the participation on the normative and the organisational part of the metamodel specification, along with the implementation of the event-management and monitoring sub-systems. [This work is summarised in Sections 5.2, 6.1 and 6.2 and Chapter 7.]

Modgil, Sanjay; Oren, Nir; Faci, Noura; Meneguzzi, Felipe; Miles, Simon and Luck, Michael (2014). "Monitoring compliance with E-contracts and norms". English. In: *Artificial Intelligence and Law* 23.2, pp. 161–196. DOI: 10.1007/s10506-015-9167-9. URL: http://link.springer.com/10.1007/s10506-015-9167-9.

This paper presents the benefits of using functional programming models in the context of the SUPERHUB project, which included the monitoring of high-level mobility policies. My contribution in this work was the participation in both 1) the design and implementation of the event modelling and disruptive event detection mechanism, and 2) the design of the mobility policy modelling and monitoring algorithm. [This work is an adaptation in the smart cities context of the contributions presented in Chapters 4, 5 and 6.]

Oliva-Felipe, Luis; Alvarez-Napagao, Sergio and Vázquez-Salceda, Javier (2012). "Towards a framework for the analysis of provenance-aware norms in complex networks ". In: Proceedings of the First International Conference on Agreement Technologies, Volume 918. URL: http://scholar.google.com/scholar?q=related:0i7FcKi2f10J:scholar.google.com/&hl=en&num=20&as_sdt=0,5.

This paper presents a proposal for combining norms and provenance concepts in order to monitor and analyse the impact of policies on tragedy-of-the-commons scenarios. My contribution was the involvement on the state of the art and on the definition of the conceptual framework. [This work is a position paper that is being developed as another PhD thesis and builds upon contributions presented in Chapters 7 and 8.]

Panagiotidi, Sofia; Alvarez-Napagao, Sergio and Vázquez-Salceda, Javier (2013). "Towards the Norm-Aware Agent: Bridging the Gap Between Deontic Specifications and Practical Mechanisms for Norm Monitoring and Norm-Aware Planning". English. In: *Coordination, Organizations, Institutions, and Norms in Agent Systems VIII*. Ed. by Tina Balke; Frank Dignum; M Birna van Riemsdijk and Amit K Chopra. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 346–363. ISBN: 978-3-642-37755-6. DOI: 10.1007/978-3-319-07314-9_19. URL: http://www.staff.science.uu.nl/~dignu101/coin2013/papers/20130207.pdf.

This paper presents a summary of some key contributions of this thesis: a solution for the fundamental distinction between norms and norm instances, and a generalisation of the formalisation and the operational semantics of norms based on my previous work. My contribution was multiple: the comparative state of the art w.r.t. compliance and verification, the explicit definition of the two layers of norm lifecycles (norm/norm instances), the formalisation of the operator and the monitoring-specific grounding of the operational semantics. The rest of the paper was a product of joint, continuous work with the rest of the authors. [This work is detailed in Chapters 6 and 7.]

Panagiotidi, Sofia; Vázquez-Salceda, Javier; Alvarez-Napagao, Sergio; Ortega-Martorell, Sandra; Willmott, Steven; Confalonieri, Roberto and Storms, Patrick (2008). "Intelligent Contracting Agents Language". In: Volume 4: Proceedings of the AISB 2008 Symposium on Behaviour Regulation in Multi-agent Systems.Aberdeen, Scotland, pp. 49–55. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=related:GdxvHmx7_OUJ:scholar.google.com/.

This paper reports the final version of the contracting language, ontological layers and communicative artifacts defined in the CONTRACT project. My contribution here focused on the state of the art that led to the conceptual framework and in the definition and implementation of the contracting language. Additionally, I also contributed to the communicative part by designing some of the contract-based interaction protocols. [This work is the basis of Section 5.1.]

Vázquez-Salceda, Javier and Alvarez-Napagao, Sergio (2009). "Using SOA Provenance to Implement Norm Enforcement in e-Institutions". English. In: *Coordination, Organizations, Institutions and Norms in Agent Systems IV*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 188–203. ISBN: 978-3-642-00442-1. DOI: 10.1007/978-3-642-00443-8_13. URL: http://www.springerlink.com/index/j37549365404x865.pdf.

This paper introduces a provenance-based norm-enforcement mechanism as an adequate framework for the monitoring of Service-Oriented Architectures. My contribution was the SOA governance part of the state of the art, and the design and implementation of the norm-enforcement architecture, including the formal translation between high-level concepts (events, regulations, protocols) into provenance assertions and production systems. [This work is used as a part of the motivation in Section 4.1, and is the basis of the architecture presented in Section 8.2.]

Vázquez-Salceda, Javier; Confalonieri, Roberto; Gómez-Sebastià, Ignasi; Storms, Patrick; Kuijpers, SP Nick and Alvarez-Napagao, Sergio (2009). "Modelling contractually-bounded interactions in the car insurance domain". In: *Proceedings of the First International ICST Conference on Digital Business (DIGIBIZ 2009)*. London.

This paper presents a use case showcasing a middleware for integrating high-level specifications of contracts in Service-Oriented Architectures resulting from the CONTRACT project. My contribution in this work consisted in the participation in the materialisation of real-world concepts into elements of the CONTRACT language, as well as the creation of relevant interaction protocols. [This work is an application of the contributions presented in Section 5.1.]

Vázquez-Salceda, Javier; Alvarez-Napagao, Sergio; Tejeda-Gómez, Arturo; Oliva-Felipe, Luis; Garcia-Gasulla, Dario; Gómez-Sebastià, Ignasi and Codina, Victor (2014). "Making Smart Cities Smarter - Using Artificial Intelligence Techniques for Smarter Mobility". In: *SMARTGREENS 2014* Proceedings of the 3rd International Conference on Smart Grids and Green IT Systems, pp. 5–13. DOI: 10.5220/0004734600050013. URL: http://dblp2.uni-trier.de/db/conf/smartgreens/smartgreens2014.

This paper presents some advances over the state of the art in smart-city management resulting from the SUPERHUB project, which included the monitoring of high-level mobility policies. My contribution in this work was the participation in both 1) the design and implementation of the event modelling and disruptive event detection mechanism, and 2) the design of the mobility policy modelling and monitoring algorithm. [This work is an adaptation in the smart cities context of the contributions presented in Chapters 4, 5 and 6.]

Other Conference and Workshop Publications

Alvarez-Napagao, Sergio; Gómez-Sebastià, Ignasi; Vázquez-Salceda, Javier and Koch, Fernando (2012). "cOncienS: Organizational Awareness in Real-Time Strategy Games". In: *Frontiers in Artificial Intelligence and Applications* Volume 220: Artificial Intelligence Research and Development, pp. 69–78. DOI: 10.3233/978-1-60750-643-0-69. URL: http://ebooks.iospress.nl/publication/6176.

This paper presents a proposal for the use of organisational structures to model concepts common to real-time strategy games, in order to provide social concepts and a higher level of adaptivity in this game genre. My contribution was the conceptual framework of the work, the state of *the art, the architecture of the proposal and the implementation of the concept in a real game (Warcraft III). [This work is a consequence and application of the contributions presented in Chapters 5, 6 and 7 and is used as a use case in Chapter 9.]*

Gómez-Sebastià, Ignasi; Alvarez-Napagao, Sergio and Vázquez-Salceda, Javier (2013). "Towards Heuristic Based Mobility Policy Optimisation". In: *Frontiers in Artificial Intelligence and Applications* Volume 256: Artificial Intelligence Research and Development, pp. 297–300. DOI: 10.3233/978-1-61499-320-9-297. URL: http://ebooks.iospress.nl/publication/35263.

This paper presents a framework for the monitoring and quantitative evaluation of high-level mobility policies. My contribution consisted in the participation on the definition of the formalisation of the normative language used to implement the policies and the formalisation of the evaluation algorithm. [This work is an adaptation in the smart cities context of the contributions presented in Chapters 5, 6 and 7.]

Gómez-Sebastià, Ignasi; Garcia-Gasulla, Dario; Alvarez-Napagao, Sergio; Vázquez-Salceda, Javier and Cortés, Ulises (2012). "Towards an implementation of a social electronic reminder for pills". In: *VII Workshop on Agents Applied inHealth Care, AHC@AAMAS2012*. Valencia, Spain. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.466.1786&rep=rep1&type=pdf#page=63.

This paper presents the results of a practical use case in which a social reminder for pills was integrated in an organisational framework for eldercare management. My contribution to this work consisted in participating on the modelling and design of the organisational and normative structures and their grounding to the ALIVE meta-model. [This work is a consequence and application of the work presented in Chapters 5, 6 and 7, and is used as a use case in Chapter 9.]

Moreno, Jonathan; Cortés, Ulises; Garcia-Gasulla, Dario; Gómez-Sebastià, Ignasi and Alvarez-Napagao, Sergio (2013). "Applying COAALAS to SPiDer". In: *Frontiers in Artificial Intelligence and Applications* Volume 256: Artificial Intelligence Research and Development, pp. 326–335. DOI: 10.3233/978-1-61499-320-9-326. URL: http://ebooks.iospress.nl/volumearticle/35269.

This paper presents the use of an organisational framework for its integration with a real system for eldercare treatment management. My involvement in this work focused on the definition of the conceptual framework and the creation of the social, organisational and normative models, and on the grounding to the ALIVE meta-model. [This work is a consequence and application of the work presented in Chapters 5, 6 and 7, and is used as a use case in Chapter 9.]

Miscellania

This PhD thesis was typeset using the LaTeX typesetting system created by Leslie Lamport and the *memoir* class, maintained by Peter R. Wilson and Lars Madsen, with MacVim, the LaTeX-BoX plugin and latexmk as the main tools. The body text is set at 11pt on a 33pc measure with the following fonts: *Athelas* (serif), *Iowan Old Style* (roman) and *Inconsolata* (typewriter). This document was compiled at December 10, 2015 at 1:43 pm.