Títol: **NORMATIVE THINKING ON WASTEWATER TREATMENT PLANTS**

Volum: **I**

Alumne: **DARIO GARCIA GASULLA**

Director/Ponent: **CLAUDIO ULISES CORTÉS GARCÍA**

Departament: **LSI**

Agraïments

Al meu director i codirector per confiar en mi i orientar-me.

A la Montse per dedicar-me el seu temps quan ni tan sols en tenia.

Als meus companys i amics per no cansar-se de preguntar-me de que anava el projecte.

A la Sara per aguantar-me i formar part d'aquest any genial.

A la meva familia. Vosaltres m'heu empés (a vegades literalment) fins aquí. Es tan merit vostre com meu.

Index

1 Introduction

1.1. Motivation and goals

This document is the report of the thesis "Normative thinking on wastewater treatment plants". This thesis was born from the interest of the author in Artificial Intelligence (A.I.). Having done all the subjects related with A.I. that the Barcelona School of Informatics (FIB) offers, I asked the teachers of my favorite ones for a thesis related with the A.I. . Ulises Cortés and Juan Carlos Nieves offered me this interesting thesis based on a doctoral thesis of environmental sciences done by Montse Aulinas [23]. The proposed work implied theoretical research, a working implementation and a real life domain to work with. I accepted without any doubt.

Aulinas's thesis proposed a multi-agent based system to manage the problems caused by the industrial wastewater discharges in rivers. She discussed that, by the use of intelligent agents in the managing process of wastewaters, there could be an important increase in the quality of the river water and in the efficiency from the organizational point of view. To do that she proposed a group of agents, which would take the roles of the most important entities in the process of wastewater discharges, from industries to the agencies in charge of controlling them, in order to represent all the involved parts. It is obvious that, for the agents to be able to work rationally, they need to interact with the laws they are subject to. That is the main issue this thesis deals with.

Based on a real world domain, this thesis proposes a way to make those laws to be comprehensible for agents. It will discuss a methodology for analyzing, specifying, implementing and testing those laws, in a generic way that can be applied to any normative environment.

The goals of this thesis are,

- To obtain a generic and complete specification syntax for analyzing laws and norms, prove that specification with an implementation of real laws applied to the given domain and
- To develop a prototype where the norms implementation can be tested using a possible real scenario.

1.2. Wastewater treatment plants (WWTP)

In today's crowded society, wastewater management is a very important issue. It directly affects the environment's pollution levels and the society's Quality of Life (QoL). To the growing production of wastewater by domestic houses, we must add the wastewater spilled by industries, which are usually mixed with the domestic ones in the sewer system.

To deal with mixed wastewaters, the Wastewater Treatment Plants (WWTP) use certain environmental policies. These policies apply to industrial and domestic entities that spill in the sewer system managed by the

receiving WWTP. Wastewater treated by WWTP ends up in a river; hence, the main objective of these environmental policies is to maintain or improve the river water quality. According to [23], there are several strategies for managing wastewater and for establishing regulations by which wastewaters can or cannot be spilled to the sewer system. As follows:

- Effluent standard based strategies: Develop regulations taking as the main consideration the emissions that will be restrained by them.
- Ambient water quality objective strategies: Establish the regulations according to certain ecosystem quality objectives.
- Economic based strategies: Define regulations which work with economic instruments such as price-based rules.

In practice, a combination of the three is the best choice. And that is the way the European Community Directives points at. The European Community Directives that deal with wastewater, such as the Directive 91/271/EEC concerning Urban Wastewater Treatment [5], sets regulations for WWTPs of all European urban settlements according to the receiving river characteristics.

1.2.1. The Catalan sanitation plan

In order to locally apply the European Directive [5], national and regional norms are being developed. In our particular case, we will work with the regional ones. The Catalan Government developed the so called Catalan Sanitation Plan that follows the European directive guidelines taking into consideration the local particularities.

The Catalan Sanitation Plan (7th November 1995) describes the quality goals for all Catalan rivers. It was developed to fulfill the European directives as well as to deal with the increasing amount of mixed urban and industrial wastewater.

All entities that spill wastewater or deal with it must comply with the Plan. Domestic and industrial spills are subject to it, as well as the WWTP and all the other entities (sewers, river, Local Water Entity and Water Catalan Agency) which interact with wastewaters at an institutional level.

Part of the Catalan Sanitation Plan [7] will be our test subject. We will take several norms from it and we will apply our norm's specification and implementation process on them. When choosing which norms are going to be used, we use the criteria of avoiding norms related with bureaucracy issues, since we want to focus on those norms directly related to river water quality. All the norms we will treat can be seen in Annex I.

1.3. Impact of norms in the WWTP management

Norms that deal with very specific issues of certain domains, such as the Catalan Sanitation Plan, are often very technical and rigid about their contents. The application of them by the obliged entities is a delicate and difficult job for the people in charge. As well as legal knowledge, technical and concrete basic knowledge are needed. Also, some special problematic features of environmental context difficult the task [24], such as:

- The facts and principles involved cannot be represented solely in terms of mathematical theory or a deterministic model.
- Imprecise and uncertain data or vagueness in the needed information.
- Huge quantity of data to analyze.

These problems added by environmental norms to WWTP management are the same ones involved in the computational solutions feasibility. It is this low-level nature of concrete environmental norms that makes them ideal candidates to be "digitized".

The formally representation of norms in a truthful and reliable way, generates diverse and very interesting advantages. In our particular domain, and just to mention some, having the current normative frame implemented in a computable way would allow a faster and more accurate way of:

- Checking their lawful state by industrial entities, before and after generating new spills.
- Studying the legality of spills by the Local Water Agency (LWA), when requests for spill authorization are received.
- Analyzing the robustness and congruency of working laws.
- Simulating the involved agent's behavior in possible real-life scenarios.
- Analyzing and filtering huge amounts of data in order to obtain only the relevant parts.

Our objective is to make all these advantages possible. And in order to do that, we need to find a way to capture norms.

1.4. Approaches for capturing norms

The main job of this thesis is to analyze how to deal with the need of specifying and implementing norms, which were originally intended for being understood and followed only by humans, in a way that can be used by artificial agents.

In order to do so we need to study what a norm is and the common and general parts of all norms. Once done that we will focus on trying to find a way to represent those parts in a computable way that can help us to achieve our goals. But first we must understand what a norm is.

1.4.1. Norms

The concept of norm is used in a lot of disciplines such as philosophy, psychology, sociology, law and computer science. Each one of those fields uses norms in a different context and with a different objective, and therefore defines it in a different way. Here we will use a definition of norm between the law one and the computer science one, as those are the fields we will be working with. We understand a norm as:

> *A rule applicable to a certain group of entities which define the behavior that is accepted by the society.*

Apart from the formal definition of it, norms in computer science are usually classified by its level of concreteness. That is, abstract norms and concrete norms [6 and 8]. Abstract norms use a very generic terminology and are thought at an institutional level, while concrete norms are specified using more particular concepts. These last ones are closer to an implementation as leave less space to interpretation.

As well as a category, norms have a lot of different attributes that, sometimes, are not formal enough or consensual enough.

Norms are usually expressed by generic locutions such as "the norm says it is obligatory" or "it is forbidden by that norm" to state what is known as *the normative content of a norm*. In more formal analysis, those contents are usually expressed as conditionals, what is known as the condition of application of the norm, to state the conditions required for the norm to be active.

If condition then consequence

This popular way of analyzing norms splits its content into two elements. The *condition* defines the requirements for the norm to apply. And the *consequence* specifies the obligations derived after the condition is executed.

As well as the body of the norm, norms usually contain the *responsibility part*, which states the agents that may be responsible for the start of end of the norm application.

All this concepts and analysis that define a norm are vague and generic, and we need a much more concrete level of analysis in order to treat them accurately. Next we will propose a system for norm analysis that will fit our needs.

1.4.2. Norms analysis

Norms, as we have seen in the previous section, are complicated and controversial. To find a way of obtaining all of their content and representing it formally is no easy task. After studying the works of several authors on the matter [3, 6, 12, 26 and 34], and our own needs when facing the specification of norms, we have decided to split the norms into four generic parts that all of them have, and which, together, represent the whole meaning of the norm:

1. **The kind of norm it is.**

   Each norm falls into one of these three categories:

   I. Norms that allow something. For example: "It is allowed to have 0.3gr/l of alcohol in blood while driving if you are a professional driver."

   II. Norms that forbid something. For example: "It is forbidden to get into a train without a valid ticket."

   III. Norms that oblige something. For example: "It is obliged to rewind the videotapes before returning them to the videotape rental store."

2. **The norm condition.**

   Each norm includes its own definition of the situation conditions under which it is applicable.

   For example, in: "If you drive at night, it is obligatory to have your car light's on." The norm conditions would be "you drive at night".

3. **The norm content.**

   Each norm states what actions and situations it refers to. We understand the content as what does the norm allow, permit or forbid.

   For example, in: "It is possible to start the final career thesis once the student has passed all the subjects of the career" the content would be "start the final career thesis".

4. **The norm subjects.**

   For each norm it is known to whom it is intended. Who or what does the norm apply to.

   For example: The norm "Drivers with less than one year of experience cannot drive faster than 80km/h" applies to "drivers with less than one year of experience".

All norms can be fully defined by those four elements or, if the norm is complex (see section 2.1), detached into simpler norms which then can be defined by those four elements.

Now that we have the norm's meaning split into four parts, the problem of formally representing norms becomes the problem of formally representing those four parts individually. The first one, the norm's kind, can be represented with deontic logic. The second and the third, the norm's conditions and content, can be represented by dynamic domain's representation approaches such as Situation Calculus. The forth and last representation, the norm's subjects, can be done in several ways depending on the final objective of the system, most of them through agent's roles.

1.4.3. Deontic logic

As we have seen in the previous point, all norms can be classified in one, and only one, of these three kinds:

- Norms that state that something is permitted.
- Norms that state that something is forbidden.
- Norms that state that something is obliged.

To represent the concepts of permission, prohibition and obligation, we must find a way of modeling those concepts formally and explicitly.

Representing those concepts is an interesting and old problem, which has been attacked since ancient Greece. The result is deontic logic [2, 14, 20, 21 and 31], a field of logic concerned specifically about those concepts which can be applied to our normative needs [8]. Although we only will use deontic logic as a modeling tool to describe the norm's kind, it is important to understand how it works and which its basic axioms are. To do so we will study the standard system of deontic logic.

On 1951, Von Wright [10] presented a formalism based on Propositional Calculus which was later reformed to use Kripke's semantics. The result is called standard system of deontic logic or KD [21].

To represent the concepts of obligation, prohibition and permission, deontic logic adds one specific operator to represent each one of them. Being the three unary operators where x is a logic formula:

- Ox to represent that x is obligatory true.
- Fx to represent that x is obligatory false.
- Px to represent that x can be true.

The formal definition of the relationship between those three operators is:

$Fx \leftrightarrow O\neg x$

"X is forbidden means it is obligatory that X is false"

$Px \leftrightarrow \neg Fx \leftrightarrow \neg O\neg x$

"X is possible means X is not forbidden" or

"X is possible means it is not obligatory that X is false"

These three operators (O, F and P) cover all the classic deontic possibilities a given formula have. A visual representation of that can be seen in figure 1.

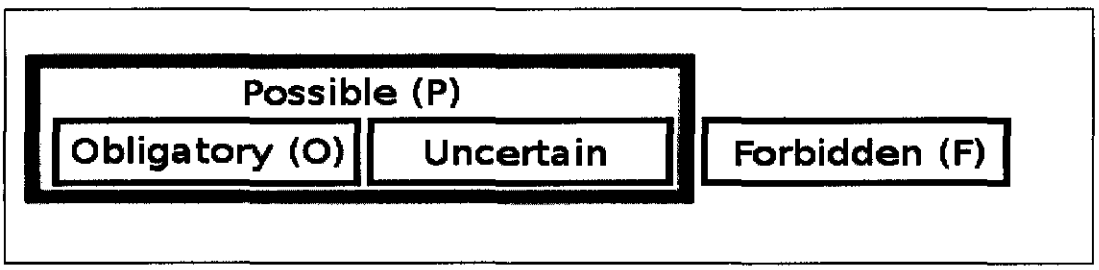

Figure 1: Threefold partition of proposition deontic value.

Classic deontic logic does not usually work with the concept of what is called "Uncertain" in figure 1, which represents those facts which cannot be proved always true neither always false. Whenever those concepts are mentioned, they are usually represented as "$P \wedge \neg O$" (possible but not obligatory).

Once we understand the basic operators and axioms of deontic logic, we can take a quick look at some of the most important axioms of KD:

$O(x \rightarrow y) \rightarrow (Ox \rightarrow Oy)$

"If it is obligatory that x implies y, then if x is obligatory, y is obligatory"

$Ox \rightarrow \neg O\neg x$

"If x is obligatory, then x is permitted (x is not forbidden)"

Several deontic formulas are derived by those axioms, but we will not get any deeper into it or KD here as it is not in the scope of this thesis and the use we will make of it is simply that of modeling the norm's deontic meaning.

1.4.4. Dynamic domains

In order to be able to represent all scenarios the norms will work with, we must analyze the domain those norms will be applied on. That means, study all the possible situations that may occur and the events that may be executed.

In the case of the WWTP, we must consider we are working with a real life domain, a domain where time is a variable on which everything depends, also known as a dynamic domain [18]. These domains are absolutely necessary to represent day-to-day concepts in a realistic way, as we intend to do, but they also bring along several problems to our solution. First of all, we must understand how a dynamic domain works.

In order to represent the world, dynamic domains work with time-lines. Basically, time-lines add time as a variable to the world. In dynamic domains, as in the real world, everything depends on when it happens. The tools used to represent that time-line concept are fluents and actions [18]. Fluents specify a certain variable which value changes through time, and actions represent events that may change the values of fluents in a certain moment.

For example: "Position" is a fluent, "Move" is an action and the action "Move" may change the value of fluent "Position". "Color" is a fluent, "Paint" is an action and "Paint" may change the value of "Color".

Following that definition, a state of the world is fully defined by the values of its fluents, which assert as true or false all the facts of the world. The relationship between the states of the world is defined by actions. As actions change the value of fluents the current situation moves from one state to another. The usual formal way of representing fluents is as logical predicates, and actions as logical functions. This will be absolutely necessary when trying to find a way of implementing them in a computable way. Once defined these two elements, fluents and actions, we will be able to represent the whole domain we are working with. In section 5.4 we will see how our specific domain will be represented in fluents and actions.

The expressiveness power that dynamic domains have must be matched with a logic paradigm able to deal with it in a computable way so that it can be put to work. In particular we will be concerned about the difficulties the concept of time-line, represented by the consideration of time as a variable of the system, adds.

Several attempts have been done to capture the concept of time-line [15 and 28] and several problems have been found along the way. The most important one is what is known as the frame problem [15], as any attempt to represent time-lines in a practicable way must solve it.

1.4.4.1 The frame problem

As we have seen, with dynamic domains we define our world by the use of actions and fluents. Fluents define values through time and actions change dynamically. This brings a huge representation power but also generates one big problem, what is known as the frame problem. And it can be easily explained like

this: It is easy to represent the effects actions have on fluents but, how do you represent the effects actions do not have on fluents?

In order to know the value of a fluent at all times we must know the effect of every action on it. Otherwise the value of a fluent after the execution of an action the effects of which have not been specified, will be unknown or, at best, an assumption.

In any domain that tries to partially represent the real world, there may be hundreds of actions and much more fluents. Stating the effects of all actions on all fluents specifically may be an almost impossible job. And if we consider that each action will affect a very low percentage of fluents, the majority of those effects will be void. An implicit way must be found to represent those "non-existing" relationships. A solution must be found to the frame problem.

There have been several attempts to solve the frame problem [4, 15, 22, 27 and 29], but we will focus on the most successful one (Situation Calculus), as it is also the one that fits our problem the most.

1.4.4.2 Situation Calculus

The Situation Calculus (McCarthy & Hayes 1969) [15] is a first order logic language designed solely to represent dynamically changing domains. To do so it uses actions and fluents, as we have defined them earlier. Situation Calculus defines the time-line as a succession of world states, linked by actions, and uses situations as the main working element. In Situation Calculus, the time-line concept is represented as a succession of situations, a history of actions occurrences that link from one to another. This way situations are not just states in Situation Calculus. Each situation is a history of events as well.

To achieve that, in Situation Calculus there are three common groups of first order logic formulas defined which can be formalized by the means of Horn clauses augmented with negation as failure:

- Action preconditions and action effects, to define in which situations every action can be executed, and once it has been done, which fluents change as a result. Each action defined has preconditions based on the situation it is executed upon. For example, before executing the action "leaveRoom(person,room)" in situation S, "personInRoom(person,room)" must be true in S. The action effects axioms can be used along with successor states axioms, but in our case we will only use the later ones as with them alone everything can be represented.

- Successor states axioms, which define all the ways a fluent value may (or may not) change. These axioms are the ones that solve the frame problem in Situation Calculus. With these axioms we will be able to fully represent the time-line concept and at the same time avoid the frame problem. These axioms are the most important part of our specification and implementation and will be analyzed deeply in sections 4 and 5.

- The foundational axioms of Situation Calculus. Those necessary to make true the Situation Calculus assumptions and some first order logic Inductions. Theses axioms formalize the Situation Calculus properties related with situations. For example, by definition two situations are the same if they are the same list of consecutive actions:

  $do(A,S)=do(A',S') \rightarrow A=A' \wedge S=S'$

  This way, even if two situations fluents are identical, the situations themselves are not equal unless the actions that lead to them are the same.

As well as those formulas, several functions are defined in Situation Calculus:

- A binary function do(A,S) is defined: action X situation → situation
  For each possible pair of action A and situation S, it returns the resultant situation of executing A in S. This function is what allows us to travel ahead in the time-line. For example: The resultant situation of executing action "paint(box,blue)" in situation "situation0" would be represented in Situation Calculus as:

  do(paint(box,blue),situation0)

- A binary function poss(A,S) is defined: action X situation
  For each possible pair of action A and situation S, it returns the viability of executing A in S. This function will be defined by the action precondition axioms, and called upon every time an action is to be performed. For example: If the action "paint(box,blue)" can be executed in situation "situation0" would be represented in Situation Calculus as:

  poss(paint(box,blue),situation0)

- A binary function holds(F,S) is defined: fluent X situation
  For each possible pair of fluent F and situation S, it returns F's truth value in S. This action will allow us to query any resultant situation, so we can explore the value of any fluents on it. For example: The value of fluent "color(box,blue)" in situation "situation0" would be represented in Situation Calculus as:

  holds(color(box,blue),situation0)

Finally, an initial situation constant is given. Usually named "s0", it represents the initial moment where the first action will occur. For s0 we will assume perfect knowledge. That is, that we know everything that is true and false in it. That will make easier some work we have to do and will be important when dealing with some logic issues as "Negation as failure" (see section 5.1.1).

Adding the initial situation to all the above defined elements, we can represent any situation of the domain. Next we will see an example of it:

In this example, the domain is defined only by two fluents and two actions:

Fluents

- *color(object,color)*
- *position(object,position)*

Actions

- *paint(object,color)*
- *move(object,initial_position,final_position)*

In our example's initial state "Situation1", the next two fluents hold:

- *color(box,blue)*
- *position(box,floor)*

This means that, in "Situation1" there is a blue box on the floor. In figure 2 we can see a graphic representation of what happens when we execute two actions: "paint(box,red)" and "move(box,floor,table)" in the initial state "Situation1".

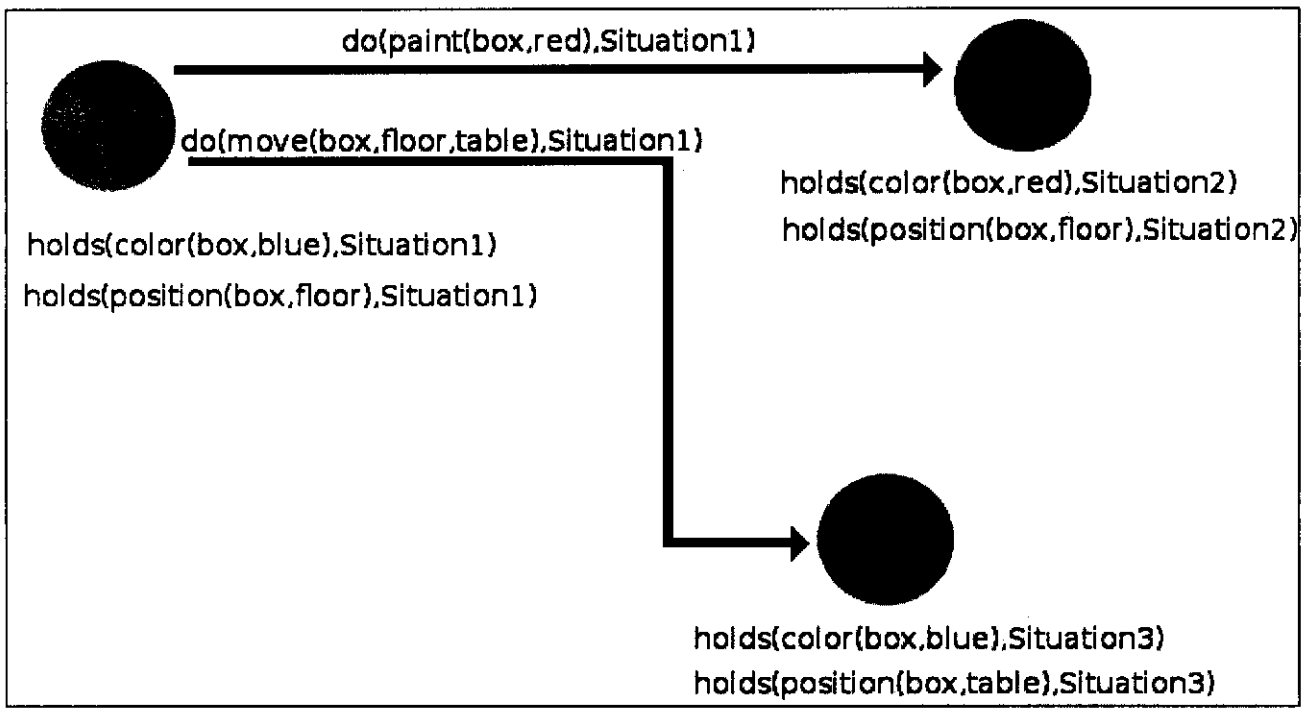

Figure 2: Example of actions and fluents use in Situation Calculus.

With Situation Calculus tools we can represent any step given in a time-line, and by a nested usage of them, we can represent as many steps as we want or need to. For example, if we want to know the value of *fluent(F)* after executing three consecutive actions, *action1(A)*, *action2(B)* and *action3(C)* starting at initial situation *s0*, it would be as simple as:

holds(fluent(F),do(action3(C),do(action2(B),do(action1(A),s0))))

Every situation can be defined as a nested and ordered succession of actions occurred from initial state. The concrete implementation of Situation Calculus formulas will be seen and explained in section 5.

1.4.5. Agent's roles

When defining the subjects of a norm, the set of entities that are obliged to its accomplishment, there are several approaches. All of them have one concept in common, the concept of role within the society [19]. At the highest level, a role is no more than a label put to those entities which share certain characteristics. In our case those characteristics will be norms that apply to them.

At specification level, agent's roles are defined by the requirements of the norm's definition. When implementing though, the roles are a more delicate thing. It must be taken into consideration the environment they will be used on and the requirements of it. Nevertheless, roles will be a necessity for us.

1.5. Plan of the document

This document is structured as follows: The first part is an introduction to the thesis content, the domain it works on and the different theoretical issues it deals with. The second part ("Approach of this thesis") is the description of the thesis approach to its different goals. The third ("Requirements analysis") is a study of the requirements of the problem; once we have seen which it is and how do we plan to solve it. The forth ("Specification of norms") explains the specification of the solution proposed, which implementation's will be see in the fifth ("Implementation"). In the sixth ("Prototype") we will see a working prototype where the work done in this thesis will be put to practice. In the seventh ("Economic analysis") we will make an economic analysis of it and in the eighth ("Conclusions and future work") we will look back at all the work done and draw several conclusions and thoughts for future work.

2 Approach of this thesis

After studying the problem we are facing, we have realized that we must take certain decisions before trying to solve it in the basic fields explained in the previous section. Next we will explain and justify some of the resolutions we have taken before starting to work on our particular problem.

2.1. Norms analysis

In section 1.4.2 we have proposed a norm decomposition, which could work in our domain and ease our work at analyzing norms. It is important to understand certain decisions we have taken about it now that we have a complete overview of the problem we are facing.

After studying the generic and common way of understanding norms as conditionals [6] it has been clear to us that, not only we do not need to use it but it would make our task more difficult.

Working with a domain specified in a dynamic logic such as Situation Calculus allows us to represent the concept of time-line in a much simpler and effective way than conditionals. The conditional approach tries to solve the problematic representation of temporariness of actions. Problem Situation Calculus solves it for us.

For example, while in conditional form would be:

If action then reaction

In Situation Calculus it is translated like:

In situation A, an action is performed, the resultant situation is B

With this, we avoid the conditionals and just need to implement the rules that define the behavior of reactions to actions with Situation Calculus axioms, a task those axioms were defined to do.

An interesting and particular case we have noted concerns "permission norms" (norms that allow something). In section 1.4.2 we stated that with our four parts decomposition, we could represent all the content of a norm. That is true for one norm, but in real life norms, sometimes a norm is more than one norm.

Allowing norms very often contain more content than the one which can be represented by a single norm. We call them "complex norms", as they can be decomposed in more than one "atomic norm". Usually, those allowing, complex norms do not only state what is permitted, but implicitly state as well what is forbidden.

For example, the norm:

It is possible to have liquids on your hand luggage when boarding into a plane if they are smaller than 100ml.

Implicitly states that:

> *It is forbidden to have liquids on your hand luggage when boarding into a plane if they are bigger than 100ml.*

If we are lucky, allowing norms will be complemented with a specific forbidding norm which unequivocally states what happens in the cases where the norm does not apply. In that case nothing special must be done. But by our experience, that will not happen often.

The best way we found to deal with those "complex" allowing norms with implicit meaning, instead of making a particular and surely more difficult analysis of them, is to split them into "atomic" norms and formalize them separately. That is the policy we will follow in the specification and implementation of norms in sections 4 and 5. A deeper analysis and the specification of that particular case of norm will be seen in section 4.2.4.

Regarding the norms subjects (the agents roles) we have found unnecessary and inefficient to specify them in their own and particular way. Once more, we will use the Situation Calculus properties to define and use them. That means defining roles as fluents of the domain, with all the associated properties. That will allow us to have an easy and fast way of consulting and modifying them. We will see in the implementation (section 5) how this will be represented.

2.2. Situation Calculus

The decision of using Situation Calculus has proven to be a very satisfactory one. It is a reliable, stable and complete way of representing dynamic domains and has successfully dealt with all its difficulties. Despite that, we have had to make some adjustments to the axioms and implementation methods usually used, in order to make them fit our particular case.

The successor states axioms will became our basic axioms. With it we will define the effect of all actions on all fluents so that we solve the frame problem and at the same time represent the temporariness of our domain. Following Reiter's solution presented in [29], in section 5 we will see an adapted implementation of those axioms specially thought to work with the specification of norms we propose in section 4. Our use of successor states axioms will make unnecessary to implement any action effect axioms. The content the action effect axioms would add to our system will be already represented by the successor states axioms.

We will also make intensive use of the binary operator "poss" to represent the action preconditions. The precondition axioms will be defined separately, and the final queries we will make to them will always be through the "poss" operator. That will make our code easier to write and to understand.

Concerning the initial situation (s0), we will be sure to fully and unequivocally define it. That means, stating all the fluents values on that situation. If the code is implemented the same way, that will make any future situation reachable from s0 fully and unequivocally defined as well. We realize that may not be a very realistic assumption as in real cases some data may be unknown, but because the main objective of this thesis is to propose and test a specification and implementation of norms, we found more important to test correctly their behavior than to simulate 100% realistically the domain performance. Moreover, in normative frameworks inaccuracy is usually not appreciated.

2.3. Prolog

As Situation Calculus axioms will all be represented by Horn clauses, the use of a logic programming language is almost mandatory. In our case we have chosen Prolog. The language, which was created in 1972 by Alain Colmerauer and Phillipe Roussel [1], has been long tested and proved efficient when dealing with logical problems.

Prolog's logic is represented by facts and rules which are a perfect fit for Situation Calculus fluents and actions. The version we will be using of Prolog (see section 3) includes negation as failure, a feature our interpretation of the domain will need. Also, Prolog's query kind of interaction will be useful to analyze the different situations reachable by our actions.

2.4. Prototype

When developing any prototype, several decisions must be done before starting. Some of them aim to help the usability and some to ease the development and help the integration of the resultant system.

For us, the first big decision must be the programming language it will be done with. In our case, we have chosen Java for several reasons. It has a large amount of libraries available for use which we might find very helpful. Among them there is a library for integrating Prolog with Java and a library to develop graphic environments. Those libraries, as we will see in section 6, will be very useful to us.

Another big decision is the kind of interface we will do. The use intended for the prototype must be taken into consideration before starting it. As our prototype is not aimed to a unique kind of user, we have decided to create a simple, generic interface. It will be easy to use, it will allow the quick generation of multiple scenarios and it will display the information in a simple, intuitive way. We intend to develop a prototype useful for as many different applications as possible. We will see the prototype in depth in section 6.

2.5. Preliminary study of the WWTP domain

At this point, we have seen how we will attempt to solve the problem we presented regarding norms, but before starting to work with norms, we must understand what we require of our domain and what does it have to offer to us.

As the domain's sole purpose is to serve correctly as a framework for the norms, its definition must be done according to the norms requirements. The only limitation will be the need of making it compatible with Situation Calculus, as it will have to be implemented with it. That means decomposing the domain in fluents and actions in order to fulfill the functionalities explained in section 1.4.4.2.

We will make here a first attempt to decompose the WWTP domain, following a non strict Situation Calculus style, in a way which can satisfy all of our norm's needs. Two lists will be made, one for the events and one for the facts, which will represent everything the norms legislate over. Everything else, everything outside the norm's representative meaning, will be unnecessary.

An example of that extraction from a pair of norms will be seen next, the full list will be found in Annex II.

2.5.1. First approach to WWTP's actions and fluents

Next we will see a norm and the list of actions and fluents of the related domain that we can extract from it.

Norm:

> *7.1 For the next agents is obligatory to obtain an authorization and to respect the restriction from Annex[1] I and Annex II:*
>
> - *Non domestic users whose activity is included in C, D and E of sections of Economic Activities Catalan Classification (Decree 97/1995) considered potential pollutant agents.*
> - *Those who generate spills > 6000 m3/year.*

Fluents:

- *spill_respects_restrictions*
- *agent_is_domestic*
- *spill_is_authorized*
- *activity_is_potential_pollutant*
- *agent_activity*
- *agent_spills_total_size*

Actions:

- *agent_make_spill*
- *agent_cancel_spill*
- *add_substance_to_spill*
- *delete_substance_from_spill*
- *set_agent_type*
- *set_agent_activity*
- *set_pollutant_activity*
- *request_authorization_to_agent*

[1] The mentioned Annex I and II refer to the Catalan Sanitation Plan Annex I and II. The first one contains a list of the limited substances and their limitations, and the second one contains a list of the forbidden substances.

- *approve_spill_authorization*
- *reject_spill_authorization*

Norm:

7.3. Only if the pertinent agent consider it is best to spill to the environment then it is possible to do not spill to the sewage system.

Fluents:

- *agent_is_pertinent_agent*
- *best_place_to_spill*
- *spill_place*

Actions:

- *set_agents_pertinent_agent*
- *set_agents_best_place_for_spill*
- *set_spill_place*

3 Requirements analysis

The requirement analysis of a software project studies the needs of the users the project is intended to and analyzes those needs in order to know what the final system must do to satisfy them. As this project aims more at the investigation of normative systems generally and in a concrete domain than at the creation of a user-orientated system (although the project does have several commercial possibilities as we have seen in section 1.3 and will result in a usable prototype) the requirements analysis will not follow the standard structure.

Instead of focusing only on what the user may need, we will also consider what the solution strategy, introduced in section 1.4 and explained in section 2, needs to be feasible and as optimal as possible. Obviously, we will also analyze the possible requirements that potential users of our final prototype may have and try to satisfy them.

3.1. Human resources requirements

When studying how to analyze norms (section 1.4) we proposed a four part decomposition of norms which, as a whole, could represent all the normative meaning of any norm. While that decomposition is and will be very useful to us, it does not magically translate real life norms into logic predicates. As the meaning of norms is very often subject to human interpretation and context specific, we need an expert on the matter the norms legislate upon (in our particular case, the WWTP) plus an expert in law (preferably in the kind of law we are working with) to interpret faithfully the meaning of those norms. Once done that, and following those two experts' directives, a logic programmer would be in charge of dismantling every norm and producing a formalization of them following the steps seen in this thesis.

For future norms decomposition and analysis it is be strongly recommended to have the experts above mentioned at the disposal of the project. In our case, we did not have such resources, which generated more work and background research by the only part involved (the author of this thesis) and several readjustments in the norms interpretation. It is necessary to bring up Montse Aulinas's invaluable help, Dr. in Environmental Sciences and expert in WWTP, in our task of understanding the normative framework and verifying certain concepts and interpretations.

3.2. Software requirements

For the development of this thesis, a sort of different software tools will be needed. First of all, a Prolog interpreter, which will be used to implement and test the Situation Calculus axioms. We have chosen SWI-Prolog [32] because it implements the standard Prolog, it has GNU license, and is one of the more popular Prolog interpreters, meaning it has been tested long and well. Any text editor will be good to program the Prolog part of the code, although it will be helpful if it can understand the Prolog syntax (as Gedit [9] does).

For the prototype development, an integrated development environment (IDE) is recommended. Considering we will use Java as programming language, we have chosen NetBeans [25]. To help us develop the graphical interface of the prototype we will use the NetBeans' GUI Builder. To integrate the Java code with the Prolog code we will use Tuprolog [33] a Java-based, light-weight Prolog. Finally, to execute the prototype we will need a copy of the Java Runtime Environment (JRE [13]).

3.3. Performance requirements

As we explained in section 2.4, our prototype will not focus on a single kind of user or a single kind of use. Therefore, analyzing the requirements will be a difficult task. Focusing on the information our prototype will offer allows us to fix several generic requirements. To start with, it has to be fast, as it will be dealing with currently ongoing scenarios. Certain decisions on sensible data may have to be taken based on its results, so liability will be a priority as well.

The prototype could have many uses, so the user knowledge could vary a lot. In order to make it intelligible for as much people as possible, it will be important to keep a simple and non-technical terminology in the parts intended for user interaction. That will have to be kept in mind since the norm analysis start.

3.4. Hardware requirements

One of the strong points of this thesis is the generation of an application which is highly usable and has a huge representation power in such a simple and powerful language as Prolog is. That will allow us to develop and run our prototype in almost any modern computer without performance issues.

Being so, for the development of this thesis and the execution of the prototype, we will only need a domestic computer powerful enough to run all the software above mentioned.

4 Specification

In section 1.4.2 we saw an analysis of norms which could allow us to represent norms formally. We also saw a study of the logic paradigm known as Situation Calculus which could allow us to represent their behavior in a dynamic domain. Now we must link those two parts together so that we can reach a functional goal.

In this section we will go through all the process we have done to specify norms. First we will analyze and try a generic formalism proposed by others. Then we will see how to link that formalism with the one we proposed in section 1.4.2. Finally we will generate a specification syntax from our formalism and see it applied to all the norms we work with.

4.1. A first approach to the specification of norms

Before trying to produce a definitive specification of norms, and considering the difficulty norm's content representation have, we have decided to take several intermediate steps in order to fully understand each norms meaning before producing the final specification. In this section we will see a formalism proposed by [26] to analyze norms and how does it apply to some of our norms.

This intermediate analysis is not absolutely necessary for the specification process, and probably it will not be done once we are familiarized with the formalizing process of norms, but considering our poor knowledge of the domain we considered recommendable to study it.

A norm is formally defined as follows:

```
norm(Id,
    (ActivCond,
    ExpirCond,
    MaintCond,
        (Op,
        RoleOrActor,
        NormContent
        )
    )
)
```

Each of those elements beeing:

- Id (Identifier)
  It is the identifier of the norm. It will be represented by a unique number.
- ActivCond (Activation Conditions)
  The activation conditions define the state conditions under which the norm is active. It will be represented as a list of facts linked by first order logic operators.
- MaintCond (Maintenance Conditions)
  The maintenance conditions define the state conditions under which the norm keeps being active, once it has been activated. It will be represented as a list of facts linked by first order logic operators.
- ExpirCond (Expiration Conditions)
  The expiration conditions define the state conditions under which the norm stops being active. It will be represented as a list of facts linked by first order logic operators.
- Op (Operator)
  The Deontic operator of the norm. O for obliged, F for forbidden, P for possible. It will be a single character.
- RoleOrActor (Role or Actor)
  The role or group of actors to which the norm applies. It will be a unique string, the group identifier.
- NormContent (Norm Content)
  The norm content specifies the state conditions to which the norm obliges, forbids of permits, once it has been activated. It will be represented as a list of facts linked by second order logic operators.

As we said before, this analysis is useful to start working with the domain if we are not familiarized with it. To do so, we studied all the selected norms (Annex I) with it. That analysis can be seen in Annex III. Nevertheless, that analysis will not be essential to the production or understanding of the final specification, reason why it will not be explicitly explained here.

With this analysis, we have seen a first approach to an applied norm formalization. In order to implement norms as we intend to do (with Situation Calculus), we will need an analysis addressed at our own direction. Like the one proposed in section 1.4.2. Next we will see a way of connecting those two similar yet different formalisms.

4.1.1. Relationship between formalisms

In the previous section we saw a syntax for norms formalization, and in section 1.4.2 we saw our own, which is based on splitting them in four elements. Next, we will see the direct relationships between the first and the second, so that we can translate norms easily from one to another. That translation process will also allow us to test our proposed syntax, and see if it contains all that it must and nothing else.

The four parts of norms as we split them in section 1.4.2 and the relationship with the seven elements of the generic formalism above explained are:

1. The kind of norm it is.
   It is represented by the element "Op", which will be a deontic operator.

2. The norm conditions
   They are represented by the joined concept of the three state conditions: "Activation condition", "Maintenance condition" and "Expiration condition". The sum of these three elements will represent all the states the norm is active in, and therefore the "norm conditions".

3. The norm content
   It is directly represented by the element "NormContent".

4. The norm subjects
   The range of individuals is stated in the element "RoleOrActor".

As we have just seen, our analysis can be directly translated from the one proposed by [26]. The only concept left out is the "Id" element, which is not present as we did not found it relevant for the norm study (being an arbitrary number as it is), but which will be present and necessary in the implementation of norms. In case of being necessary at this moment, a fifth element "Id" could be easily added to our specification.

Knowing that, we can continue working only with our syntax towards the specification of norms. Next we will study it and how to get a specification out of it.

4.2. Proposed specification

In this section we will extract a specification from the analysis proposed in section 1.4.2. That specification will be done particularly for norms and having in mind the logic formalism proposed for the solution (Situation Calculus). Finally we will specify the selected norms of the Catalan Sanitation Plan (Annex I) with it.

To produce the specification, we will study each of the four parts of our analysis on its own, and in the end we will give a joined specification which will represent the whole norm specification. The first and the last element of the analysis (the norm's kind and subject) do not entail much difficulty so we will deal with them first.

4.2.1. Study of the norm's parts

As we saw in section 1.4.3, the first part we identified of a norm was "the kind of norm", which will be represented by a deontic operator. In our specification it will be represented with a capital letter indicating if it is an obliging norm (O), a forbidding norm (F) or a permitting norm (P). No further specification will be needed for it as it holds no further meaning about our norms. Its only peculiarities will be related with the implementation, and will be seen in section 5.3.

In section 2.1 we saw that the forth element of our analysis, "the norm subjects", which could be represented by roles, would not be much of a problem either. Regarding the specification of roles, they will be represented by a simple label set by the norms requirements. This way we will have "the norms subjects" in the terms of the rest of the norm, as normal fluent, circumstance that will make their implementation and integration in the rest of the domain very easy.

The two remaining elements to be specified, "the norm condition" and "the norm content", will need a deeper study as they contain the temporariness of norms and, by extension, the real difficulty of the system.

If we remember the "norm condition" from section 1.4.2, it defines the situation requirements which determine the active or inactive state of a certain norm. That is why we will call its resultant specification the "activation states".

About the "norm content", it defines the situations every norm rules upon (the kind of rule depends on the kind of norm) or in other words, the situation requirements which determine the respected or violated state of a certain norm. That is why we will call its resultant specification the "violation states". Next we will see our approach to the specification of both.

4.2.2. Dynamic activation state specification

Finding a way to specify the situations where a norm is valid is no easy task, especially because it may contain and contribute to the norm temporariness, the main difficulty we are facing.

Instead of focusing on a static specification of the activation states, where only the required characteristics for activation would be set, we will add to the specification the dynamic element, the time-line. That will complicate the resultant specification, but will simplify the final implementation step. Specifically we will include in the specification the actions that may activate it and the actions that may deactivate it, elements that let us play with the concepts of "before" and "after" when talking about the activation state.

To sum up, in order of defining the activation state of a norm we will analyze two elements:

- The active/inactive state of the norm in a certain moment (static element).
- The effects a certain action may have on that state in a future moment (dynamic element).

For the first one, the activation state of a norm in a given moment, we will follow what we saw in section 4.1.1. That simplified syntax of characteristics will be easily translated in the implementation into Situation Calculus by the use of fluent following Horn clauses syntax.

For analyzing the actions that affect a norm, the second element, we will group those actions based on the kind of effect they have on the activation state. The resultant categorization will be:

- The actions that may activate the norm or "activation actions".
- The actions that may deactivate the norm or "termination actions".

Knowing the requirements of the activation state in a given moment (first, static element), and knowing the fluents that may be modified by every action (obtained from the study of the actions and fluent of the domain seen in Annex II), we can obtain the second element, a list for every norm of the actions that may activate it and one of the actions that may deactivate it.

Because of the frame problem, it is important to fully specify all norms. That is, list all the possible actions that may change in any possible situation the state of the norm. No matter how unlikely or strange may those situations be. Each of the actions included in those both lists will have attached information. The state conditions required for its effect to be operative (it is obvious that the resultant effect of the action "sit down" depends dramatically on our position). That information will be added to the specification for each action.

The problem of associating each action with the situation characteristics that make it effectively affect the norm activation state (formally linking the two elements of this part's specification) will be dealt with in the implementation section.

Once knowing how to specify correctly the state of a norm in a certain moment and the activation and termination actions, we can fully specify any "norm's condition".

4.2.3. Violation state specification

The violation state shares all the basic characteristics with the activation state. Both contain a situation definition (this one set the norm content while the other one set the norm conditionals). Both have a binary "state" (the first one is active/inactive and this one is violated/respected). And the state of both may be modified by the effect of actions. That is why it is possible to deal with the violation state specification in exactly the same way as we did with the activation state.

As following the same procedure which has already been done would add no information about the performance of our proposal or the possibilities of normative behavior in dynamic contexts, we decided to try a different approach for it. Instead of setting the dynamic characteristic integrated in it, as we did with the activation state (specifying the actions within the norm condition), we decided to leave it out of the violation state specification.

We will propose a static specification of the violation state, while the temporariness of the "norm's content" will be represented by the domain itself. The interesting part will be to see if a static violation state, which has been done in accordance with the domain, can work and keep the temporariness of the system when set into a dynamic context.

That static specification will be very similar to the static element of the activation state specification. Basically it will contain a logic representation of the situation requirements, by the means of Horn clauses formed by domain's fluents. An example can be seen in section 4.4.

4.2.4. Final specification

From the first analysis of norms we saw in section 4.1 to our final proposal in this section, we have learned several things. It is obvious now that the first approach proposed by [26] was not optimal when dealing with the temporal logic we intend to use, and we consider this last specification syntax a simpler and more complete solution for us. The specific reasons for this will be explained in section 8.2.

Our final proposed specification contains the next elements for each norm:

- A deontic operator stating the kind of norm it is. O, F or P.
- A definition of the state conditions needed for the norm to be active.
- The "activation actions". A list of the actions that may activate the norm and the conditions needed for each one.
- The "termination actions". A list of the actions that may deactivate the norm and the conditions needed for each one.
- A definition of the state charactersitics the norm's content rules upon.

- A role identifier, representing all those entities subject to the norm.

This specification does not have to be taken as a definitive one, especially considering we have chosen two different approaches for the "norm's content" and the "norm's condition". Once we know the performance of both solutions, in section 8.2, we will discuss which could be a formal and definitive specification for norms. In section 4.4 we will see this one applied, the one we will use for our implementation, to the norms we work with.

4.3. Allowing norms

Before seeing the final specification, a particular case must be studied, as it will imply specification particularities. That is the case of "allowing norms".

Following the interpretation we made of norms, we considered that what is allowed cannot be violated. At the same time we realized that part of the allowing norm's content was not represented only by an allowing norm, and we assumed that most allowing norms contained a forbidding norm within.

On section 2 we saw the example of the norm:

> *It is possible to have liquids on your hand luggage when boarding into a plane if they are smaller than 100ml.*

That norm implicitly states that:

> *It is forbidden to have liquids on your hand luggage when boarding into a plane if they are bigger than 100ml.*

That is caused by the ambiguous meaning and use of the word "possible" in human language. Following deontic logic (section 1.4.3), the operator "possible" can be defined as "not forbidden". Applying that direct translation, the resultant norm would state that:

> *It is not forbidden to have liquids on your hand luggage when boarding into a plane if they are smaller than 100ml.*

That would not solve our problem, since it would not legislate upon the cases not applied by the original norm. The question is then, what happens when the liquids are bigger than 100m? The original norm does not explicitly state anything about it, but it is obvious for any human reader that the norm does implicitly legislate upon that situation as well, as it would not make sense otherwise.

As a consideration, if the norm had been expressed in another way:

> *It is forbidden to have liquids on your hand luggage when boarding into a place if they are bigger than 100ml.*

The problem would be solved, as the cases not specified are considered not forbidden not obligated, and therefore need no legislation. This could be easily solved with a more unequivocal edition of norms.

Going back to our problem, in order to formalize the cases with implicit meaning, it will be necessary to explicitly state that implicit content. Our solution to do so will be splitting those allowing norms in two, one

which cannot be violated and just states what can be done (allowing part), and one which can be violated and states what cannot be done (forbidding part). Those two norms legislate upon the same content and are complementary. They together represent all the possible situations regarding the scenario ruled by the norm. For example, splitting the above seen norm, the allowing part would be:

> *It is possible to have liquids on your hand luggage when boarding into a plane if they are smaller than 100ml.*

And the forbidding part would be:

> *It is forbidden to have liquids on your hand luggage when boarding into a plane if they are bigger than 100ml.*

Those two norms together completely state what happens when you board a plane with liquids following the original norm's meaning.

To make that feasible, their activation states must be dependent on each other. That is, the allowing will be active when the forbidding is not and vice-versa. One of the norms must always be active and both must never be at the same time. To simplify and assure that, all the elements related with the activation condition will be defined in only one of the two norms. The other norm's activation state will be defined upon the activation state of the first one. An example of that will be seen next, in section 4.4.2.

4.4. WWTP norms specification

We have seen the norms we work with in section 1.2.1 and we have explained and justified a specification of norms all along this section. Finally we can put it all together and specify the selected norms (Annex I) using it.

In order to ease the norm's specification understanding, we will see first the norm in its original form, and then the norm specification, following the 4.2.4 schema. Only two norm's specifications will be seen here, the rest are in Annex IV.

4.4.1. Norm 7.1

Original norm

> *For the next agents is obligatory to obtain an authorization and to respect the restrictions of Annex[1] I and II:*
>
> - *Non domestic users whose activity is included in C, D and E sections of Economic Activities Catalan Classification (Decree 97/1995) considered potential pollutant agents.*
> - *Those who generate spills > 6.000 m3/year.*

[1] The mentioned Annex I and II refer to the Catalan Sanitation Plan Annex I and II. The first one contains a list of the limited substances and their limitations, and the second one contains a list of the forbidden substances.

Norm specification

Operator: O

State conditions for activation:

(domestic_agent $\wedge$ potential_pollutant_agent) $\vee$ m3_generated_by_agent > 6000

Activation actions:

- make a spill.
  Activates norm if: With that spill the agent generates more than 6000 m3/year.
- add a substance to a spill.
  Activates norm if: With the added quantity the agent generates more than 6000 m3/year.
- set the type of the agent to non domestic.
  Activates norm if: The agent was potential pollutant.
- set the agent activity to one considered pollutant.
  Activates norm if: The agent was domestic.
- set an activity to be considered pollutant.
  Activates norm if: The agent was domestic and it does that activity.

Termination actions:

- set the type of the agent to non "non domestic".
  Terminates norm if: The agent generates spills < 6000 m3/year.
- set the agent activity to one not considered pollutant.
  Terminates norm if: The agent generates spills < 6000 m3/year.
- set the agents current activity to non pollutant.
  Terminates norm if: The agent generates spills < 6000 m3/year.
- cancel a spill.
  Terminates norm if: The agent spills quantity becomes less than 6000 and the agent is not "non domestic" or is not a pollutant agent.
- delete a substance from a spill.
  Terminates norm if: The agent spills quantity becomes less than 6000 and the agent is not "non domestic" or is not a pollutant agent.

Norm content:

Obtain an authorization and respect the restrictions of Annex[1] I and II.

[1] The mentioned Annex I and II refer to the Catalan Sanitation Plan Annex I and II. The first one contains a list of the limited substances and their limitations, and the second one contains a list of the forbidden substances.

4.4.2. Norm 7.3

The norm 7.3 is an allowing norm. In section 4.3 we discussed about the particularities of that kind of norm. Now we will see those issues applied to our specification.

Original norm

Only if the pertinent agent consider it is best to spill to the environment then it is possible to do not spill to the sewage system.

Norm specification

Norm 731

Operator: F (Forbidden)

State conditions for activation:

Norm 732 not active

Norm content:

Spill in the sewage system.

Norm 732

Operator: P (Possible)

State conditions for activation:

The pertinent agent considers best to spill in the environment.

Activation actions:

- An agent changes the place it considers best to spill.

  Activates norm if: The agent is the pertinent agent and the place is the environment.

- The agent changes its pertinent agent.

  Activates norm if: The new pertinent agent considers best place the environment.

Termination actions:

- An agent changes the place it considers best to spill.

  Terminates norm if: The agent is the pertinent agent and the new place is not the environment.

- The agent changes its pertinent agent.

  Terminates norm if: The new pertinent agent do not consider best to spill in the environment.

Norm content:

Spill in the sewage system.

5 Implementation

During the first four points of this document we have analyzed and theorized about normative systems and how to bring them to a real application. All that work was done with the sole intention of establishing a solid background which could allow us to develop a reliable and meditated implementation of the norms applicable in our case. And that last step is what we will do in this section. It is important to note that without the previous four sections, this solid and efficient implementation could not have been done.

The implementation we will see here will not only be that of the norms, as that would make an isolated normative system impossible to test. We will as well implement the domain itself, following the same principles used for the norms, around which the norms work. We are proud to say that most of the work done in the specification can and will be applied to the domain implementation (as will be seen in the next sections). That itself speaks for the versatility and robustness of the solution proposed.

We will start by talking about the decisions taken and studying the choices we had. Once knowing where we are, we will focus on the how to get where we want to go. In the second part we propose an implementation schema (probably the most notable part of this thesis), which will be the base of our implementation. Finally we will see the resultant code of both the WWTP domain and the related norms.

5.1. Implementation decisions

Before jumping straight into the implementation of norms, a lot of decisions must be taken. It is absolutely necessary to state a standard implementation method and follow it all along the development, if we intend the system to be robust and coherent.

As the implementation has been entirely done in Prolog using the Situation Calculus formalism (for reasons explained in section 1 and 2), many of those decision will have to do with concepts related with them. Opposite to Situation Calculus, for which we have given a general overview, we will assume some knowledge on Prolog's basic way of working (for example, Prolog's unification algorithm [16]) as it would be long and unnecessary to explain it here.

We will study as well some key issues related to implementation norms, which had no space in the specification section due to their low-level nature, but which are a necessity for the viable performance of the normative framework.

5.1.1. Prolog and Situation Calculus

We will start this approach to the implementation by analyzing some basic concepts of Prolog and Situation Calculus. Some of those concepts where explained in section 1.4.4.2 but will be briefly reminded here.

5.1.1.1 Prolog basic operators

In order to facilitate the reader's understanding of the coming Prolog code, and assuming some knowledge on Prolog's unification algorithm, we will introduce here some of the most common and basic operators used in Prolog. Most of them are intuitive.

- *A :- B*

  If B is true then A is true.

- .

  End of rule.

- ,

  Logic conjunctive separator of terms.

- ;

  Logic disjunctive separator of terms.

- *A = B*

  True if A and B have the same value.

- *A \= B*

  True if A and B do not have the same value.

- *\+ A*

  True if A cannot be proved true (NAF see section 5.1.1.4).

- *Variable*

  All items starting with a capital letter have no fixed value. Prolog's unification algorithm will try to give them one that makes the rule containing them true. All apparitions of a variable in a rule share the same value.

- *atom*

  All items starting with a non-capital letter are fixed values.

5.1.1.2 Holds and poss

We briefly introduced situation calculus functions in section 1.4.4.2. Here we will just remind what we already saw.

Situation Calculus defines the present moment (situation) as a succession of actions, with the execution of which the reality changes. To know what is true in a given situation, Situation Calculus uses a predicate

named "holds". "holds" sole function is to state the truth value of a given fluent in a given situation. Its formal definition is:

holds(F,S): fluent X situation

For each possible pair of fluent F and situation S, holds returns F's truth value in S.

To know when an action is executable in a situation, that is, to help simplify things and join the action preconditions check in a single predicate, "poss" is created. "poss" function is to tell us if an action can be executed in a situation. Its formal definition is:

poss(A,S): action X situation

For each possible pair of action A and situation S, it returns the viability of executing A in S. For example, to execute the action of deleting an amount X of substance Sub from a spill Sp it is necessary that:

- *The spill Sp exists*
- *The substance Sub exists*
- *Quantity X is not negative*
- *The spill Sp contains at least X amount of substance Sub*

Which translates into code Prolog with the predicate "poss" as:

```
poss(del_substance(Sp,Sub,X),S):-

holds(spill(Sp),S),

holds(substance(Sub),S),

X>=0,

holds(substance_spill_quantity(Sp,Sub,Y),S), X=<Y.
```

During the development of the specification and implementation, we have realized that fluents and norms are very similar. Both have binary states (active/inactive and violated/respected for norms, true/false for fluents) related with a certain situation and the states of both are modified by the execution of actions. Being so, we have decided to treat norms and fluents in a very similar way. Having as we do a predicate (holds) to know when a fluent is true, we can use it as well to know when a norm condition is "true". We will therefore use holds to know when a norm is active in a situation and we will create an implementation schema based on holds which can be applied to both. This decision is more significant than it looks like. The conclusions achieved and the repercussions of it will be discussed in section 8.2.

Consequently, in order to represent the "dynamic activation state" of a norm we will use "holds", following the typical Situation Calculus syntax.

holds(N,S): norm X situation

For each possible pair of norm N and situation S, holds returns F's truth value in S.

We will not make a similar analogy with "poss" as norms do not have preconditions and therefore do not need of it.

Norms and fluents have big similarities as we have seen, but they also have differences. Norms have what we call a "violation state", a list of situation conditions under which its content is not respected. That concept do not exist for fluents and that is why we decided to add a predicate named "violated" to those defined by Situation Calculus.

5.1.1.3 Violated

Apart from the predicates included by Situation Calculus treatment of actions and fluents, norm's content will need one more. An element to represent when a norm is not being respected is needed. Situation Calculus do not have any tool predefined for it as it is a concept not found in fluents neither actions. We define it formally as follows:

violated(N,S): norm X situation

For each possible pair of norm N and situation S, violated returns if the norm's content is being respected (true) or not (false).

This predicate will contain the "norms kind" and "the norms content" seen in our first analysis of norms. When consulted it must return the violation state, regardless of the norm being an obliging or a forbidding one (the differences between both will be dealt with within the implementation of the violation state). About allowing norms (P) it is important to note that they will not have this predicate defined, as their content cannot be violated.

When defining the function "violated" another decision was taken. We assumed that a norm can only be violated when it is active. This may look as an obvious observation, but it brings consequences. The good side is that it will save us from checking the activation state of a norm separately from the violation state whenever we want to know if the norm is being respected. To know if the norm is violated we will just ask if the violation predicate is true, and it will only be able to return true if the activation state is true.

The problem with that decision is the limitations it adds. With that definition of "violated only if active" it will be impossible for us to know if a norm is violated when it is not active. The usability of that information depends on the domain and the objective of the system. In our case we considered unnecessary to know it, and considered the advantages more relevant than the disadvantages. We understand though that in some other domains or solution approaches it could be useful. The code modifications needed to change that decision are not difficult to do, but require specific knowledge of our approach.

Next we will see a key issue regarding Prolog. It is the treatment of negation as failure.

5.1.1.4 Negation as failure

One of the reasons we had for using Prolog as our logic programming language was that it included negation as failure. Negation as failure or NAF comes from the derivation of "not P" from failure to derivate "P", that is "if we cannot prove P true, P is proved false".

Following our decision of assuming perfect knowledge about the domain information (represented by the initial state s0 knowledge, explained in section 1.4.4.2), comes the fact that everything which cannot be proved true is false. That is why NAF will not only be useful, it will be commendatory. That will be seen in how the Situation Calculus predicates will work.

5.1.2. Norm's variables

Regarding norms, we have discussed long about them. We have deeply analyzed them, we have taken decisions and we justified them. But when facing the last step of the formalization of norms, one last issue arises, the variables related to each norm.

If we remember the norm's content when we first analyzed it, it was represented by a list of predicates linked by first and second order operators. For example:

> *It is obligatory to obtain an authorization and to respect the restriction from Annex[1] I and Annex II*

Can be translated into logic as:

$$\forall Spill\ Sp: Sp\_done\_by\_agent \Rightarrow ($$
$$S\_authorized\_by\_associated\_entity \wedge$$
$$\forall Substance\ Sub: Sp\_contains\_Sub \Rightarrow (Sub\_respects\_limitation \wedge Sub\_not\_forbidden)$$
$$)$$

In order to represent that with Prolog's logic predicates, we would have to define fluents using lists (the most complicated data structure Prolog can handle) as parameters. Those lists would contain the elements for which the fluent was true and the members of those lists would be modified by the actions, which would add or delete them.

Needless to say, that is very complicated and would require of complex recursive functions, difficult to test, for each required list handling method. Using recursive functions in a dynamic context where every action and fluent depends on the time-line is no triviality. Luckily for us, there is a way of avoiding it; defining the norms and fluents to refer to a single instance of the elements they rule upon, instead of a list of them. Instead of having a predicate ruling upon twenty instances, we will have twenty predicates ruling upon one instance. For example, instead of:

> *pollutant_substances(List_of_substances)*

We will have:

> *pollutant_substance(Single_substance)*

[1] The mentioned Annex I and II refer to the Catalan Sanitation Plan Annex I and II. The first one contains a list of the limited substances and their limitations, and the second one contains a list of the forbidden substances.

That will simplify our code, and the interaction with it, but will multiply the number of the predicates generated by Situation Calculus. As we are working with Prolog, which unification algorithm is extremely efficient, we will not have performance issues related with the amount of predicates generated.

What we will call the "norm's variables" is then a list of variables for which a norm has an activation state. If a norm legislates upon what an entity can do, then an entity will be its only variable. If it legislates upon what an entity spill must satisfy, then both entity and spill will be the variables. As we do not work with lists but with individual variables, an instance must exist for every possible combination of values for which every norm is applyiable.

As a particular case and in order to identify them separately, we will add one more variable to a norm. It's identifier. Instead of defining a predicate for every norm we have, we will use the same header "norm" to represent the activation state of all norms, and the first parameter "IdNorm" to identify them. All those variables will be represented in the header of each norm, following the Prolog syntax. For example, for the norm 7.1:

> *For every agent, it is obligatory to obtain an authorization and to respect the restriction from Annex[1] I and Annex II*

The generic Prolog header will be:

> *norm(71,IdAgent)*

While the particular rule referring to norm "71" and agent "ag1" will be:

> *norm(71,ag1)*

With this, all our fluents and norms will be reduced to true or false predicates by means of adding all the possible parameters for each one. This will make our implementation simpler to understand and easier to work with.

5.2. Implementation schema for the activation state

As we said in the previous section, we will deal with norms and fluents almost identically. In fact, we will consider norms as extended fluents, which can be violated. The common part will therefore be the "activation state", the other half of the implementation of a norm. Norm's "dynamic activation state" and fluents current value are analog and can be implemented using the same schema. And that is what we want to prove here.

To implement them we will use a quite rigid syntax, one which solves the frame problem and gives us the maximum utility possible. It will follow Reiter's simple solution to the frame problem [29], and we consider it a practical example of it adapted to normative frameworks.

To start with, we must understand how we pretend to solve the frame problem. The idea is very simple. The frame problem is the difficulty of explicitly stating the inexistent effects of most of the actions of a domain on

[1] The mentioned Annex I and II refer to the Catalan Sanitation Plan Annex I and II. The first one contains a list of the limited substances and their limitations, and the second one contains a list of the forbidden substances.

most of its fluents. To do so, we will explicitly state all the existent effects of all the actions of the domain on the fluents, and assume all the non-explicitly stated actions are inexistent. To sum up:

If we state all that happens, what we do not state, does not happen.

Following that idea we identify the three only possible cases where a norm is active:

- The norm was not active, the last action could activate it and it did so.
- The norm was active, the last action could deactivate it but it did not.
- The norm was active and the last action could not deactivate it.

With that idea in mind we produce the next implementation schema, which can be translated directly to Situation Calculus.

We can assert that a norm "N" is active after doing an action "A" in situation "S" ("holds(N,do(A,S)" is true) if and only if:

- N was not active, A may activate N, and the conditions needed for A to activate N are true in S.
  We will call this the **activation cases**.
- N was active, A may deactivate N but the conditions needed are not true in S.
  We will call this **maintenance cases**.
- N was active, A may not deactivate N.
  We will call this non **termination cases**.

With these three elements we have fully and only represented the states in which the norm is active. There is no other way for the norm to be active. In all those states that are not represented by any of those three scenarios, the norm is not active.

Looking at it, we can clearly see the relationship between those cases and the specification parts we worked on before. The actions which could activate or deactivate every norm where already identified in the specification (see sections 4.2.2 to 4.2.4). We named them "activation actions" and "termination actions". We saw that they were necessary for the specification of norms, and now we have seen they are as well necessary for its implementation. The "activation actions" will be used in the "activation cases" and the "termination actions" will be used in the "maintenance cases".

In a similar way, the "conditions needed", which are mentioned in the "activation cases" and the "maintenance cases", were also stated in the same sections (4.2.2 to 4.2.4), in relation with every affecting action. For every activating action we had the activation conditions, and for every termination actions we had the termination conditions.

As we can see, the specification proposed in 4.2 defines perfectly everything that we will need to implement norms.

5.3. Implementation schema for the violation state

As we discussed in section 4.2.3, we will treat violation states in a very different way than we did with the activation state. As we will leave the dynamic component out of it, the violation state implementation will be much simpler because it will be static.

It is important to remember as well our decision of including the activation state within the violation state. That is, a norm can only be violated if it is active. That requirement will be visible in the implementation.

Violation state does have one minor difficulty in relation with the activation state, its behavior changes depending on the deontic operator of the norm, while the activation state behavior is independent of the norms kind. We must then give one definition for each of the two possible behaviors it may have: Obligation and forbidding. And implement each norm depending on its kind (O or F). We will define them as follows.

We can assert that a norm N is violated in situation S if and only if:

- The norm is active in situation S, the norm obliges to the value of one or more fluents and S does not fulfill all of those obliged fluents.
- The norm is active in situation S, the norm forbids the value of one or more fluents and S fulfills one of those forbidden fluents.

As we discussed before, allowing norms (P) cannot be violated, so the previous implementation covers all the possible for norm's violation state. Adding that to the implementation schema for activation state, we now have a way of formally representing norms and putting them to work. That is what we will do in the last sections of this thesis.

5.4. Domain implementation

In section 1.4.4.2, we saw how Situation Calculus uses fluents and actions to represent all that happens and is true in a domain. In section 5.1 we saw that fluents and activation state of norms are very similar, to the point that they can be implemented using the same tools. Finally, in section 5.2 we saw a schema to implement norms.

In order of being consequent with all that has been said and to prove it, we will use the same schema proposed for norms, to implement the fluents of the domain. We will just adapt its terminology to one closer to fluents, which will make much clearer the similarities with Reiter's work [29].

We can assert that a fluent F is true after doing an Action A in situation S if and only if:

- F was not true, A may activate F, and the conditions needed for A to activate F are true in S.
- F was true, A may deactivate F but the conditions needed are not true in S.
- F was true and A may not deactivate F.

As well as with norms, those three scenarios cover all the possibilities for a fluent to be true. The fluents will be false in any other case.

The only restriction that specification generates is the need of making all fluents work in an active/inactive way, like norms do. We already explained the issues that arise, and the fact that it does not affect us negatively in section 5.1.2.

Regarding actions, the only explicit definition we will give of them is the already explained predicate "poss". This predicate will define the precondition requirements for the action to be runnable. The rest of its definition will come implicit in the effect every action will have on all fluents and norms. Every action will be defined by its repercussion in the world.

5.5. WWTP norms implementation

In this section we will see Prolog code according to Situation Calculus formalism, intended to represent norm's content in a formal way, following all the above discussed. As we have seen we will split norm's implementation in two: holds predicate and violated predicate. We will take a look first at the predicate that defines the activation state of norms, "holds", and then to the predicate that defines the violation state of norms, "violated".

5.5.1. Norm's holds implementation

Due to the size of this part of the implementation, here we will only see, detailed and commented, one norm's holds implementation. The rest is in Annex V, with the rest of the Prolog code.

5.5.1.1 Norm 7.1.

In this example we will use the norm 7.1., which states:

Is obligatory to obtain an authorization and to respect the restrictions of Annex[1] I and II:

- *For non domestic users considered potential pollutant agents.*
- *Those who generate spills > 6.000 m3/year.*

In section 4.2.1 we identified the activation conditions as the last two lines of the norm. We also listed the actions that could affect the activation state of this norm:

Actions that may activate it:

- make_spill
- add_substance
- set_agent_type
- set_agent_activity
- set_pollutant_activity

[1] The mentioned Annex I and II refer to the Catalan Sanitation Plan Annex I and II. The first one contains a list of the limited substances and their limitations, and the second one contains a list of the forbidden substances.

And the actions that may deactivate it:

- set_agent_type
- set_agent_activity
- unset_pollutant_activity
- cancel_spill
- del_substance
- del_total_substance

Now let's see the implementation once applied those two lists to our schema (section 5.2). Everything after "//" is a comment:

```
1  //Norm header
2  holds(norm(71,IdAgent),do(A,S)):-
3  //Begin "The actions that may activate the norm, when the conditions for it are satisfied"
4  //If the norm is inactive, the action is possible and with the new spill the total size of spills is bigger than 6000, make a
5  spill activates the norm for this agent.
6  A = make_spill(IdSpill,IdAgent) , holds(spill_total_size(IdSpill,SizeS),S) ,
7  holds(agent_spills_total_size(IdAgent,SizeA),S) , Total is SizeA+SizeS , Total>6000,\+holds(norm(71,IdAgent),S)
8  , poss(A,S) ;
9
10 //If the norm is inactive, the action is possible and with the added quantity the total size of spills is bigger than 6000, add
11 substance activates the norm for this agent.
12 A = add_substance(IdSpill,Sub,Qua) , holds(agent_spill(IdAgent,IdSpill),S) ,
13 holds(agent_spills_total_size(IdAgent,Size),S) , 6000<Qua+Size ,\+holds(norm(71,IdAgent),S) , poss(A,S);
14
15 //If the norm is inactive, the action is possible and the agent is pollutant, set the agent type to non_domestic activates
16 the norm for this agent.
17 A = set_agent_type(IdAgent,non_domestic) , holds(pollutant_agent(IdAgent),S) , \+holds(norm(71,IdAgent),S)
18 , poss(A,S);
19
20 //If the norm is inactive, the action is possible and an activity is pollutant, set that one as agent's activity activates the
21 norm for this agent.
22 A = set_agent_activity(IdAgent,Activity) , holds(pollutant_activity(Activity),S) ,
23 holds(agent_type(IdAgent,non_domestic),S) , \+holds(norm(71,IdAgent),S) , poss(A,S);
```

```
24
25  //If the norm is inactive, the action is possible and the agent's type is non_domestic, set the agent's activity as pollutant
26  activates the norm for this agent.
27  A = set_pollutant_activity(Activity) , holds(agent_activity(IdAgent,Activity),S) ,
28  holds(agent_type(IdAgent,non_domestic),S) , \+holds(norm(71,IdAgent),S) , poss(A,S);
29  //End "The actions that may activate the norm, when the conditions for it are satisfied"
30  //Begin "The actions that may deactivate the norm, when the conditions for it are not satisfied"
31  //If the norm is active and the action is possible, set the type to non_domestic does not deactivate the norm for this
32  agent.
33  holds(norm(71,IdAgent),S) , A = set_agent_type(IdAgent,non_domestic) , poss(A,S);
34
35  //If the norm is active, the action is possible and the spills are bigger than 6000, set the type of the agent to one different
36  than non_domestic does not deactivate the norm for this agent.
37  holds(norm(71,IdAgent),S) , A = set_agent_type(IdAgent,AgentType) , AgentType\=non_domestic ,
38  holds(agent_spills_total_size(IdAgent,SizeA),S) , SizeA>6000 , poss(A,S);
39
40  //If the norm is active, the action is possible and an activity is pollutant, set the agents activity to that one does not
41  deactivate the norm for this agent.
42  holds(norm(71,IdAgent),S) , A = set_agent_activity(IdAgent,Activity) , holds(pollutant_activity(Activity),S) ,
43  poss(A,S);
44
45  //If the norm is active, the action is possible and the spills are bigger than 6000, set the agents activity to one that is not
46  pollutant does not deactivate the norm for this agent.
47  holds(norm(71,IdAgent),S) , A = set_agent_activity(IdAgent,Activity) , \+holds(pollutant_activity(Activity),S) ,
48  holds(agent_spills_total_size(IdAgent,SizeA),S) , SizeA>6000 , poss(A,S);
49
50  //If the norm is active, the action is possible and the spills are bigger than 6000, set the agents activity to non pollutant
51  does not deactivate the norm for this agent.
52  holds(norm(71,IdAgent),S) , A = unset_pollutant_activity(Activity) , holds(agent_activity(IdAgent,Activity),S) ,
53  holds(agent_spills_total_size(IdAgent,SizeA),S) , SizeA>6000 , poss(A,S);
54
55  //If the norm is active, the action is possible and the activity is not that of the agent, set an activity to non pollutant does
56  not deactivate the norm for this agent.
```

```
57  holds(norm(71,IdAgent),S) , A = unset_pollutant_activity(Activity) , \+holds(agent_activity(IdAgent,Activity),S)
58  , poss(A,S);
59
60  //If the norm is active, the action is possible and the agent is pollutant and non_domestic, canceling a spill does not
61  deactivate the norm for this agent.
62  holds(norm(71,IdAgent),S) , A = cancel_spill(IdSpill,IdAgent) , holds(agent_type(IdAgent,non_domestic),S) ,
63  holds(pollutant_agent(IdAgent),S) , poss(A,S);
64
65  //If the norm is active, the action is possible and without the cancelled spill the agent spills more than 6000, canceling a
66  spill does not deactivate the norm for this agent.
67  holds(norm(71,IdAgent),S) , A = cancel_spill(IdSpill,IdAgent) , holds(spill_total_size(IdSpill,SizeS),S) ,
68  holds(agent_spills_total_size(IdAgent,SizeA),S) , SizeA -SizeS > 6000 , poss(A,S);
69
70  //If the norm is active, the action is possible and the spill does not belong to the agent, deleting substance from a spill
71  does not deactivate the norm for this agent.
72  holds(norm(71,IdAgent),S) , A = del_substance(IdSpill,Sub,Qu) , \+holds(agent_spill(IdAgent,IdSpill),S) ,
73  poss(A,S);
74
75  //If the norm is active, the action is possible and the agent is pollutant and non_domestic, deleting a substance from one
76  of the agent's spills does not deactivate the norm for this agent.
77  holds(norm(71,IdAgent),S) , A = del_substance(IdSpill,Sub,Qu) ,holds(agent_spill(IdAgent,IdSpill),S) ,
78  holds(agent_type(IdAgent,non_domestic),S) , holds(pollutant_agent(IdAgent),S) , poss(A,S);
79
80  //If the norm is active, the action is possible and without the deleted amount the agent spills more than 6000, deleting an
81  amount from one of the agent's spills does not deactivate the norm for this agent.
82  holds(norm(71,IdAgent),S) , A = del_substance(IdSpill,Sub,Qu) , holds(agent_spill(IdAgent,IdSpill),S) ,
83  holds(agent_spills_total_size(IdAgent,SizeA),S) , SizeA -Qu > 6000 , poss(A,S);
84
85  //If the norm is active, the action is possible and the spill does not belong to the agent, deleting all of a substance from a
86  spill does not deactivate the norm for this agent.
87  holds(norm(71,IdAgent),S) , A = del_total_substance(IdSpill,Sub) , \+holds(agent_spill(IdAgent,IdSpill),S) ,
88  poss(A,S);
89
90  //If the norm is active, the action is possible and the agent is pollutant and non_domestic, deleting all of a substance
91  from one of the agent's spills does not deactivate the norm for this agent.
```

```
92  holds(norm(71,IdAgent),S) , A = del_total_substance(IdSpill,Sub) , holds(agent_spill(IdAgent,IdSpill),S) ,
93  holds(agent_type(IdAgent,non_domestic),S) , holds(pollutant_agent(IdAgent),S) , poss(A,S);
94
95  //If the norm is active, the action is possible and without a substance the agent spills more than 6000, deleting all of that
96  substance does not deactivate the norm for this agent.
97  holds(norm(71,IdAgent),S) , A = del_total_substance(IdSpill,Sub) ,holds(agent_spill(IdAgent,IdSpill),S) ,
98  holds(agent_spills_total_size(IdAgent,SizeA),S) , holds(spill_total_size(IdSpill,Qu),S) , SizeA -Qu > 6000 ,
99  poss(A,S);
100 //End "The actions that may deactivate the norm, when the conditions for it are not satisfied"
101 //Begin "The actions that may not deactivate the norm"
102 //If the norm is active, the action is possible and it is not any of those above used, the action does not deactivate the
103 norm for this agent.
104 holds(norm(71,IdAgent),S) ,\+ A = set_agent_type(IdAgent,Type) , \+ A = set_agent_activity(IdAgent,Activity) ,
105 \+A = unset_pollutant_activity(Activity2) ,\+ A = del_total_substance(IdSpill,Sub) ,
106 \+A=del_substance(IdSpill,Sub,Qu) , \+ A = delete_agent(IdAgent) ,\+ A = cancel_spill(IdSpill,IdAgent) ,
107 poss(A,S).
    //End "The actions that may not deactivate the norm"
```

At first sight, the implementation may look too big to be easy to understand, but it is just the opposite. In order to make it as clear as possible, all the variables that may affect the norm are set, and the meaning of every clause is unique and unequivocal.

If we analyze the first part, "the actions that may activate the norm when the conditions for it are satisfied" (lines 3 to 29), we can see two common parts for each action; the requirement for the norm of being inactive before executing the action (*\+holds(norm(71,IdAgent),S)*) and the requirement of the action to be possible in the situation(*poss(A,S)*).

We can as well see there is only one rule for every action that may activate the norm. That is because in this particular case no action may activate the norm in two different ways. The only case where we could need to identify two different "activation cases" for the same action would be if there was more than one way of doing so with that action.

In the second part, "the actions that may deactivate the norm, when the conditions for it are not satisfied" (lines 30 to 100), there are two common parts as well in each rule; one to state that the norm is active prior to the execution of the action (*holds(norm(71,IdAgent),S)*), and one to state that it is possible to execute the action in that situation (*poss(A,S)*).

In this part there are a lot of rules for each action. That is so because we must specify, for every action which may deactivate the norm, all the scenarios where it does not do it. For example:

After executing the action "del_substance" (lines 72 to 83), being the norm 71 previously active, it will continue being active in three scenarios:

- When we delete a substance from a spill that does not belong to the actual agent.
- When we delete a substance from a spill of the agent and the agent is non domestic and pollutant.
- When we delete a substance from a spill of the agent and after deleting it the agent's spills are bigger than 6000.

The piece of code which states that is:

- holds(norm(71,IdAgent),S), A = del_substance(IdSpill,Sub,Qu), \+holds(agent_spill(IdAgent,IdSpill),S), poss(A,S);
- holds(norm(71,IdAgent),S), A = del_substance(IdSpill,Sub,Qu),holds(agent_spill(IdAgent,IdSpill),S), holds(agent_type(IdAgent,non_domestic),S),holds(pollutant_agent(IdAgent),S), poss(A,S);
- holds(norm(71,IdAgent),S), A = del_substance(IdSpill,Sub,Qu),holds(agent_spill(IdAgent,IdSpill),S), holds(agent_spills_total_size(IdAgent,SizeA),S), SizeA -Qu > 6000, poss(A,S);

Now we can clearly see that, even though it is a very long and specific implementation, it is quite easy to understand and even to read. Understanding how the domain and the norms work by reading the code, having minimum knowledge of the formalism used, is feasible.

5.5.2. Norm's violated implementation

Here we will see detailed and commented two norm's implementation, one for each type (O and F). The rest is in Annex V, with the rest of the Prolog code.

5.5.2.1 Norm 71

In this example we will see norm 71 which states:

Is obligatory to obtain an authorization and to respect the restrictions of Annex[1] I and II:

- *For non domestic users considered potential pollutant agents.*
- *Those who generate spills > 6.000 m3/year.*

As we have seen before, the violation state is represented in our specification by the norm's content. When we specified the norm in section 4.4.1 we saw that its content was the first line, as the two last points define the norm condition (what defines activation state). Following that specification, the norm's content states:

Obtain an authorization and to respect the restrictions of Annex I and II:

[1] The mentioned Annex I and II refer to the Catalan Sanitation Plan Annex I and II. The first one contains a list of the limited substances and their limitations, and the second one contains a list of the forbidden substances.

Clearly, the norm's type is: *O (Obligatory)*

Following the obligatory implementation schema for violation states, we can separate two scenarios in which the norm 71 is violated by an agent *IdAgent* in a situation *S*.

- When the norm is active for *IdAgent* and the agent has done a spill which violates a limitation.
- When the norm is active for *IdAgent* and the agent makes a spill which is not authorized by the entity associated to that agent.

The resultant Prolog code is:

```
1  //Header
2  violated(norm(71,IdAgent),S) :-
3  //Rule 1
4  //If the norm is active and a spill of the agent violates a limitation of a substance, the norm is violated by that agent.
5  holds(norm(71,IdAgent),S),holds(agent_spill(IdAgent,IdSpill),S),
6  holds(spill_violates_limitation(IdSpill,Substance),S);
7  //Rule 2
8  //If the norm is active and a spill of the agent is done in a place for which the agent's associated entity has not given an
9  authorization, the norm is violated by that agent.
10 holds(norm(71,IdAgent),S),holds(agent_spill(IdAgent,IdSpill),S),
11 holds(spill_place(IdAgent,SpillPlace),S),holds(agent_associated_entity(IdAgent,IdAgentEntity),S),
12 \+holds(spill_authorized(IdSpill,SpillPlace,IdAgent,IdAgentEntity),S) .
```

The implementation of violation states is much shorter than the activation states because it works only on the current state, for the reasons explained in section 4.2.3. . That means it only has to deal with the fluents which are true in the current situation, and it can ignore the actions that may lead to that situation.

The two scenarios mentioned before where the norm was violated are clearly separated in the implementation. As it was an obliging norm, if any of them is not respected, the norm is violated. The only similarity between both cases is that they need the norm to be active in the situation (*holds(norm(71,IdAgent),S)*).

5.5.2.2 Norm 73

In this example we will see the violation state of norm 73, which states:

> *Only if the pertinent agent consider it is best to spill to the environment then it is possible to do not spill to the sewage system.*

That norm is an allowing norm and following our special specification for allowing norms (see section 4.3) we split it in two norms, a forbidding one (norm 731) and an allowing one (732). We saw that an allowing norm cannot be violated and therefore the violation state of norm 73 will be the violation state of norm 731.

Norm's 731 content states:

Spill in the sewage system

And its type is: *F (Forbidden)*

Following the forbidden implementation schema for violation states, there is only one scenario in which the norm 731 is violated by an agent *IdAgent*'s spill *IdSpill* in a situation *S*.

- When the norm is active for *IdAgent* and *IdSpill*, and *IdSpill* is done in the sewage system.

The resultant Prolog code is:

```
1  //Header
2  violated(norm(731,IdAgent,IdSpill),S) :-
3  //Rule 1
4  //If the norm is active and a the agent's spill is not done in the sewer system, the norm is violated by that agent and that
5  spill.
6  holds(norm(731,IdAgent,IdSpill),S), holds(spill_place(IdSpill,SpillPlace),S), SpillPlace\=sewer_system.
```

6 Prototype

The main objective of this thesis was to analyze and implement norms formally so that they can be used by artificial agents. At this point, that objective has been achieved. Our work though, would be incomplete without a working environment where those norms can be put to test, and where our thesis can be proved.

Our objective in this section is to develop a working prototype which can be used to simulate any scenario in which the norms we have implemented are contained. Most of the decisions about the prototype have already been taken and justified. Among them was the kind of prototype. In section 2.4 we decided to make an all-purpose one. Not having a single kind of user as target, it will be usable for more objectives this way.

Concerning more technical issues, we decided to use Java as programming language. It's versatility, easiness of use and large amount of libraries being its primary reasons. The rest of libraries and tools used for its development can be consulted in section 3. We will not explain the development of the prototype code, as we consider it not to be the matter of this thesis nor interesting enough. What we will see and explain is its interface and way of use, as it contains a lot of elements only seen in theory all along this thesis put into practice and it will help the comprehension of the capabilities and limitations of what we are doing. We will also see an example of execution, in order to test the prototype.

In order to understand how does the prototype work, we must first learn some basic ideas related with the system internal and initial situations. Those are described next.

6.1. Prototype understanding

Our prototype loads an initial situation of the domain on start. Through its interface the prototype allows the user to add and sequentially execute actions on that state. The user will specify which actions, in which order and with which parameters happen in the initial situation. Those actions will modify the situation of the world (its fluents) and the state of the domain's norms (activation and violation state). The system will display those changes in an easily readable way to the user, so it can know the future effects of the actions chosen.

The initial situation loaded should vary for every use, depending on the intention of it. For example, an industrial spill producer who wants to know the potential effects of future actions would set the initial state to fit its current state of spills. Then on that state it could simulate the result of generating new spills or adding substances to those already existent. In our case we set an initial state which would help the test and understanding of the prototype.

6.2. Interface description

Our prototype has a unique screen with all the available information and actions on it. That screen can be seen in figure 3.

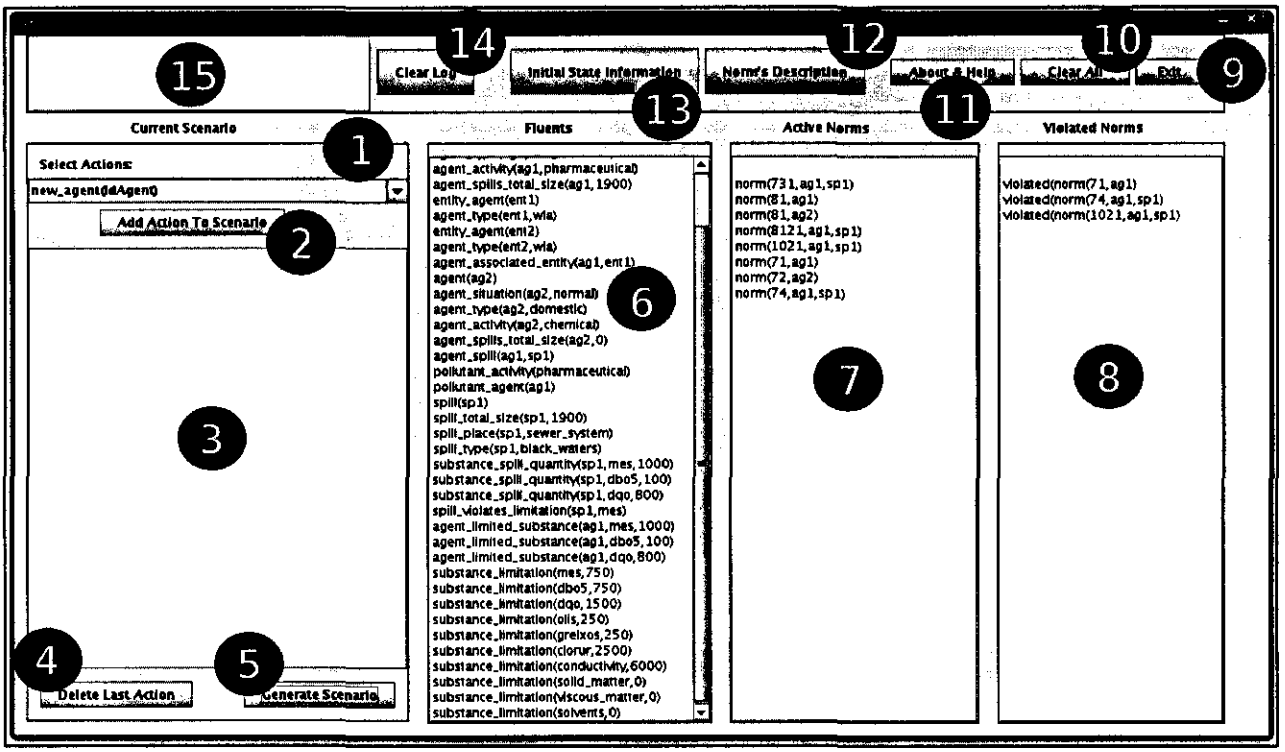

Figure 3: Prototype interface elements. Initial situation loaded.

In figure 3 we can see that the interface is made mainly of two parts. The first one contains the elements (1), (2), (3), (4) and (5) which contain the user input elements. These will allow the user to interact with the system by modifying the initial situation.

The other part contains the elements (6), (7) and (8) and represents the displaying part of the interface. Once the user has chosen the scenario it wants to test, those lists will show to the user all the associated information of the resultant state.

The rest of the elements, (9), (10), (11), (12), (13), (14) and (15) contribute with several additional functionalities to the prototype.

The detailed description of all elements is as follows:

1. **Select Actions**: Drop-down list.

Contains all the actions available in our domain. Those actions can be chosen by the user (left click), and added to the scenario by clicking *Add Action* (2).

2. **Add Action**: Button.

   On click the user will be requested to specify the parameters of the action selected on *Select Actions* (1). Once done, the action will be added to *Current Scenario* (3).

3. **Current Scenario**: Display list.

   Contains the actions chosen by the user to be executed sequentially (starting on the top of the list) on the initial state. This list of actions will define the resultant scenario.

4. **Delete Last Action**: Button.

   On click deletes the last action added to *Current Scenario* (3).

5. **Generate Scenario**: Button.

   On click generates the desired scenario which is defined by the actions listed in the *Current Scenario* (3). This button may change the value of the items (6), (7) and (8).

6. **Fluents**: Display list.

   Contains all the fluents (or facts) which are true (or hold) in the situation resultant of sequentially executing the actions listed in the *Current Scenario* (3). *Generate Scenario* (5) must be clicked to update its content.

7. **Active Norms**: Display list.

   Contains all the norms which activation states are satisfied and therefore are active in the situation resultant of sequentially executing the actions listed in the *Current Scenario* (3). *Generate Scenario* (5) must be clicked to update its content.

8. **Violated Norms**: Display list.

   Contains all the norms which contents are not respected and therefore are violated in the situation resultant of sequentially executing the actions listed in the *Current Scenario* (3). *Generate Scenario* (5) must be clicked to update its content.

9. **Exit**: Button.

   On click closes the application.

10. **Clear All**: Button.

    On click empties all the information added by the user (lists (3), (6), (7), (8) and (15)) and returns to the initial situation.

11. **About & Help**: Button.

    On click shows some information about the developer, the application and how to make it run.

12. **Norm's Description**: Button.

    On click shows information about the domain's norm ruling the domain. It shows how many there are and their original content.

13. **Initial State Information**: Button.

    On click shows the detailed information of the initial state set in the prototype.

14. **Clear Log**: Button.

    On click clears the *Log* (15).

15. **Log**: Display area.
    Shows some useful information during the execution of the prototype, such as user input errors.

6.3. Execution tests

Once familiarized with the interface and its elements, we can run it and test its performance. We will try several executions and see the resultant information given by the prototype.

6.3.1. Initial situation for the test

To ease the testing of our prototype, we have defined an initial state which contains the most common elements and will save us from creating an interesting scenario.

In the initial situation there are:

- Two spill producing agents, *ag1* of type *non_domestic* and *pharmaceutical* activity (which is considered pollutant) and *ag2*, of type *domestic* and *chemical* activity.
- Two entities agents, *ent1* and *ent2*. The first one, *ent1*, is the associated entity of agent *ag1*. *ag2* does not have associated entity.
- Most of the substances usually found on spills and its established limitations, such as: *dbo5* (750), *dqo* (1500), *mes* (750), *oils* (250), *greases* (250), *clorur* (2500), *solid matter* (0), *viscous matter* (0), *solvents* (0), etc.
- One spill *sp1* done by the agent *ag1*, of the type *black_waters* done in the *sewer_system*. It contains three limited substances *dbo5*, *dqo* and *mes*. The limitations of the first two substances are respected, the limitation of *mes* (750) is violated (1000).
- Other information required such as: industrial activities, spill places, spill types, etc.

A look of that initial situation can be seen in the figure 3 (in section 6.2). In that figure and regarding norms we can see that:

- 7 norms are activated. Norms 731, 74, 8121 and 1021 for the spill *sp1* of agent *ag1*. Norm 71 and 81 for agent *ag1*. Norms 72 and 81 for agent *ag2*.
- 3 norms are violated. 71 by the agent *ag1*. 74 and 1021, both by the spill *sp1* done by agent *ag1*.

The first violated norm, 71, states that:

> *For the next agents is obligatory to obtain an authorization and to respect the restrictions of Annex[1] I and II:*
>
> - *Non domestic users whose activity is included in C, D and E sections of Economic Activities Catalan Classification (Decree 97/1995) considered potential pollutant agents.*
> - *Those who generate spills > 6.000 m3/year.*

Agent *ag1* is *non domestic*, and as its activity (*pharmaceutical*) is considered pollutant, *ag1* is considered a pollutant agent. That makes the norm active. At the same time, the spill *sp1* is not authorized and it does not respect the substance restrictions. The norm 71 is violated by *ag1*.

The second violated norm, 74, states that:

> *If new spill then the pertinent agent will register it in a census (article 18)*

Agent *ag1* has a new spill *sp1* which is not registered. The norm 74 is violated by *ag1* and *sp1*.

The third violated norm, 1021, states that:

> *If you have the authorization then you can spill black waters to the public sewer system according to the regulations established.*

The spill *sp1* of agent *ag1* is of type *black_waters*. It does not respect the regulations established for the limited substances and it is not authorized. The norm 1021 is violated by *ag1* and *sp1*.

6.3.2. Solving violations

Once seen the initial domain and normative state, we will try to execute the pertinent actions on that situation in order to fix the happening violations.

The violated norm (74) requires the spill *sp1* to be registered in the census by an agent, but not every agent will do. In order to respect that norm, the one registering the spill must be the associated entity to the spilling agent. As the spilling agent is *ag1*, we need the agent *ent1* to register the spill *sp1* in the census (we can see that in the fluent *agent_associated_entity(ag1,ent1)* in the initial situation).

[1] The mentioned Annex I and II refer to the Catalan Sanitation Plan Annex I and II. The first one contains a list of the limited substances and their limitations, and the second one contains a list of the forbidden substances.

The action we will perform is:

register_spill(sp1,census,ent1)

By the execution of this action the agent *ent1* registers the spill *sp1* in the census. The resultant situation can be seen in figure 4.

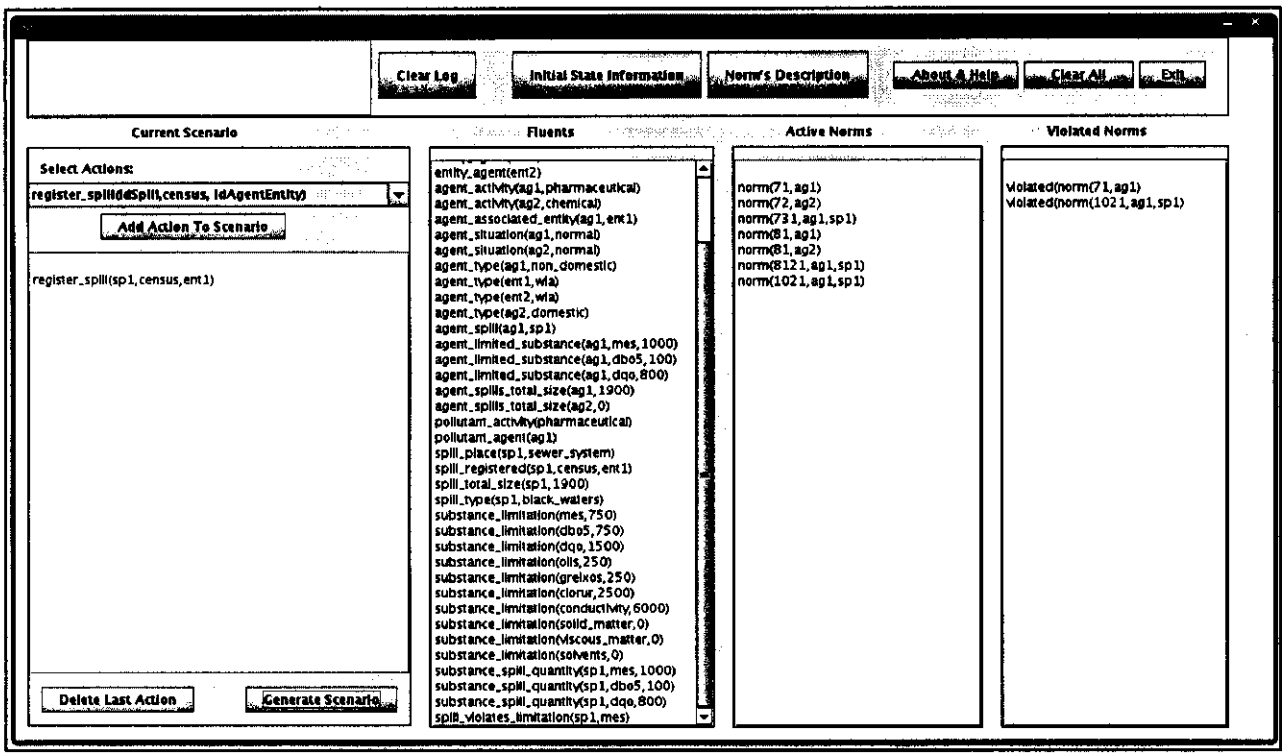

Figure 4: Prototype state after the execution of action register_spill(sp1,census,ent1) in the initial situation.

In figure 4 we can see that norm 74 is no longer violated (*Violated Norms* list). In the fluents list it can also be seen that the spill is now registered (*spill_registered(sp1,census,ent1)* by the agent *ent1*. If we take a close look we can also see that norm 74 is no longer active. That is because once a spill has been registered it cannot be unregistered, and therefore norm 74 will never apply to a registered spill. After that, only two norms remain violated, 71 and 1021.

To solve violation of norm 71 we can try a different approach. We will deactivate the norm for agent *ag1*, instead of trying to respect its content, as we know that an agent cannot violate a norm which does not apply to it. To do so, and looking at the norms definition, we can change the agent's pollutant state. *Ag1* is considered a pollutant agent, as we can see in the fluents list (*pollutant_agent(ag1)*). If we change the pollutant state of pharmaceutical activity, agent *ag1*, whose activity is pharmaceutical, will cease being a pollutant agent and the norm 71 will stop being active for it.

The action we will perform is:

unset_pollutant_activity(pharmaceutical)

This action changes the pollutant state of the activity pharmaceutical, from pollutant to non pollutant, which implicitly changes the pollutant state of all the agents whose activity is pharmaceutical. The resultant situation can be seen in figure 5.

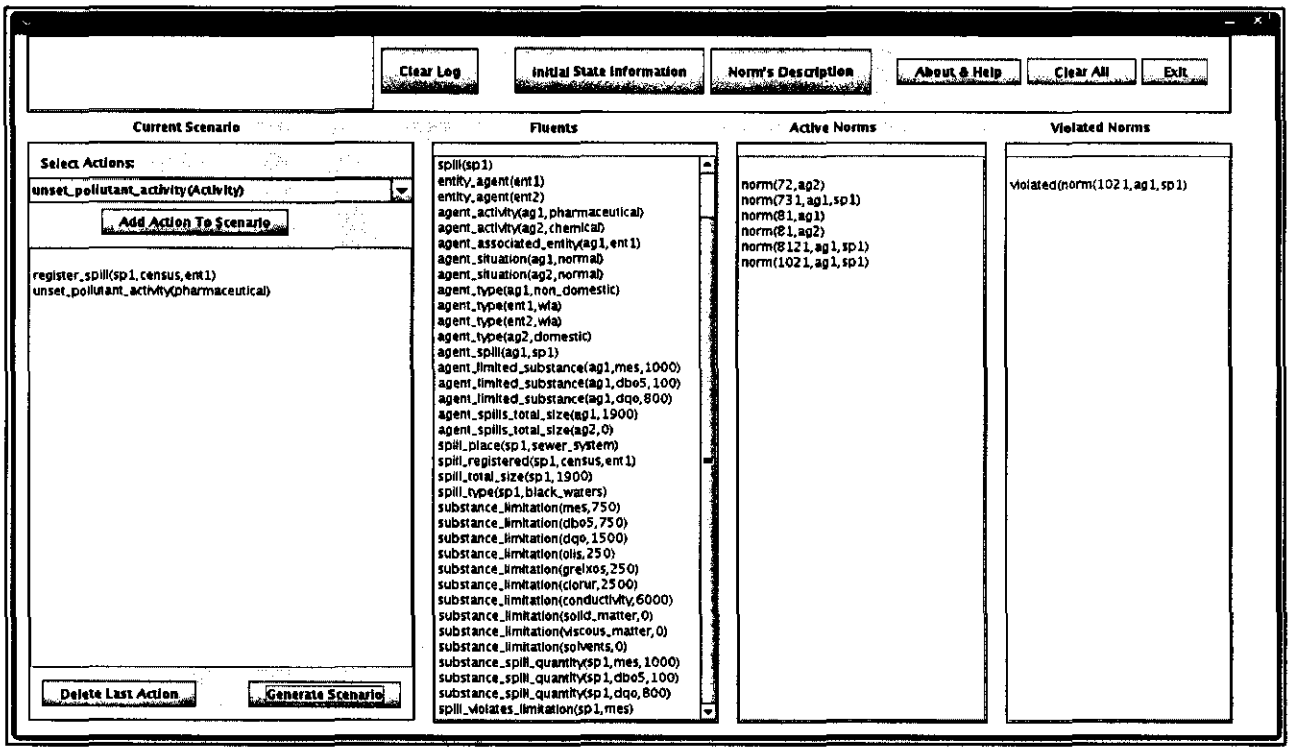

Figure 5: Prototype state after the execution of action unset_pollutant_activity(pharmaceutical).

In figure 5 we can see that norm 71 is no longer active (*Active Norms* list). As it is no longer active, it cannot be violated either (*Violated Norms* list). The effect of the action can also be seen in the fluents list, where the fluents *pollutant_activity(pharmaceutical)* and *pollutant_agent(ag1)*, have also disappeared.

Finally, to solve the last remaining violation, that of norm 1021, we will first try to make *sp1* respect the substance limitations in order to follow the established regulations. That was part of the norm's content and should be respected if we want to solve the violation. If we look at spill *sp1*, the only violated limitation is that of substance *mes*. *Sp1* spills 1000 of *mes* and the limit is 750. In the fluents list that can be seen in *spill_violates_limitation(sp1,mes)* and *substance_spill_quantity(sp1,mes,1000)*.

The action we will perform is:

del_substance(sp1,mes,300)

This action deletes 300 m3/year of mes from spill *sp1*. The resultant situation can be seen in figure 6.

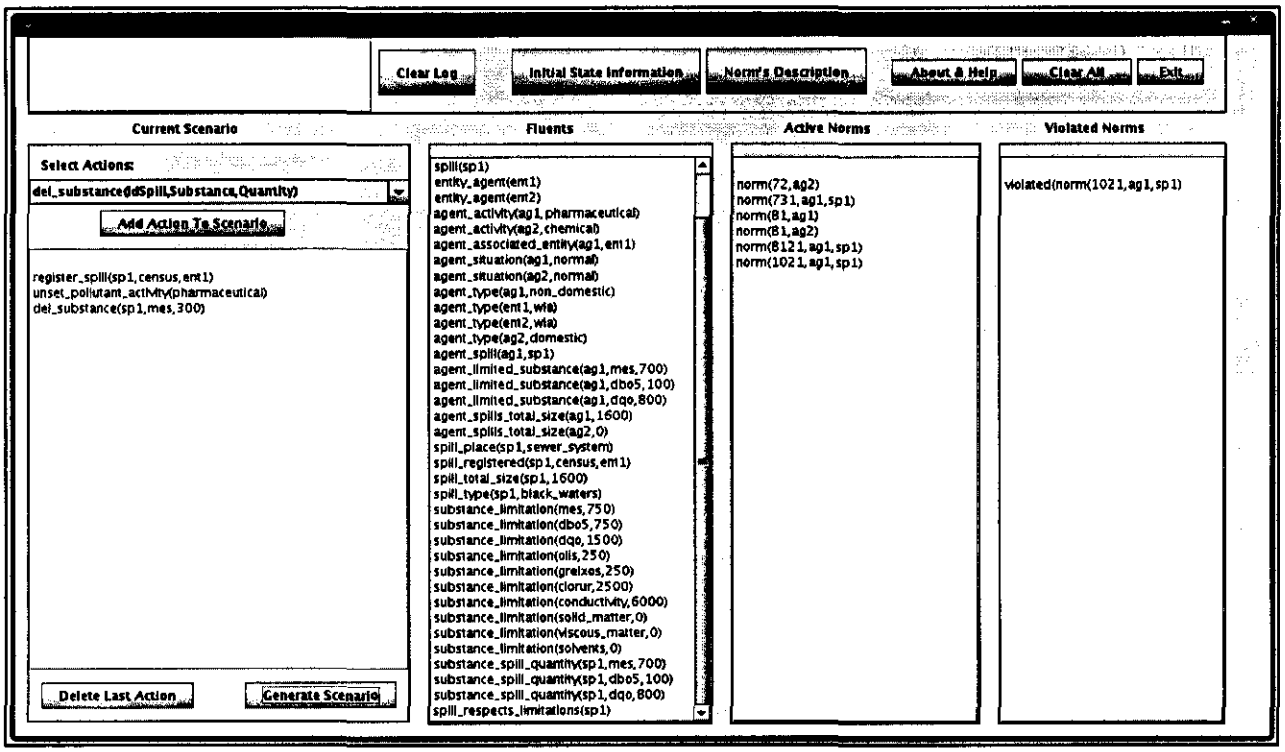

Figure 6: Prototype state after the execution of action del_substance(sp1,mes,300).

In figure 6 we can see the effects of action *del_substance(sp1,mes,300)*. In the fluents list the spill (*substance_spill_quantity(sp1,mes,700)*, *spill_respects_limitations(sp1)*) and therefore the agent (*agent_limited_substance(ag1,mes,700)*) no longer violate the *mes* substance limitation. Nevertheless, the norm 1021 is still violated. If we look at the definition of it, we will see the spill needs, apart from respecting the substance limitation, to be authorized by the entity agent; otherwise we cannot spill black waters to the sewer system. As the agent making the spill is *ag1* and its entity agent is *ent1*, the authorization to make the spill must be done by that entity agent.

The action we will perform is:

set_spill_authorization(sp1,ag1,ent1,positive,sewer_sytem)

With this action, entity agent *ent1* sets a positive authorization so that agent *ag1* is allowed to spill *sp1* in the specified place, the *sewer system*. The resultant and final situation after the execution of that action can be seen in figure 7.

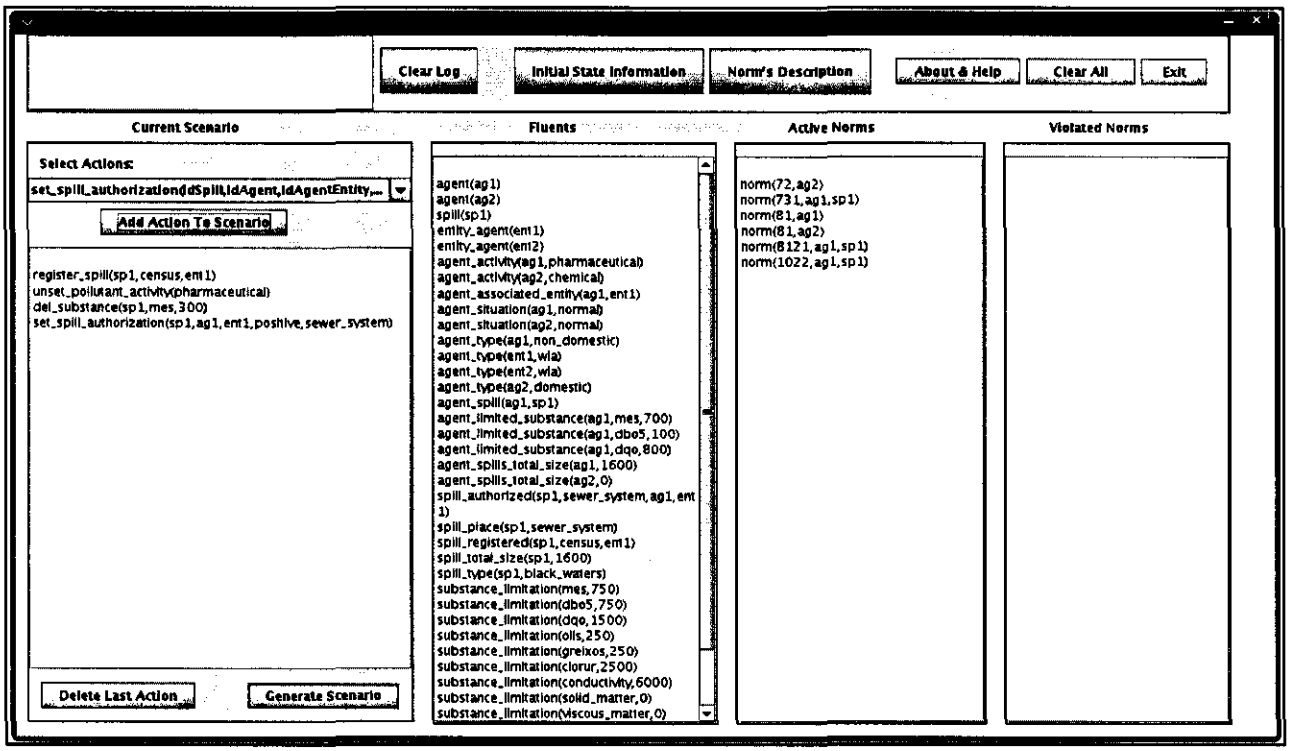

Figure 7: State after the execution of action set_spill_authorization(sp1,ag1,ent1,positive,sewer_system).

Finally, in figure 7 we can see that no norm is violated. Setting the positive authorization by the right agent finally respected the norm's 1021 content.

In this execution example we have seen how, after the sequential execution of the four actions:

- *register_spill(sp1,census,ent1)*
- *unset_pollutant_activity(pharmaceutical)*
- *del_substance(sp1,mes,300)*
- *set_spill_authorization(sp1,ag1,ent1,positive,sewer_system)*

We managed to modify the world's state in order to respect several norms.

With this example we have seen that, ruled by the sequential execution of actions introduced by the user, this prototype allows us to affect the domain's state (represented by fluents) and see its evolution through time. It also shows us how our norm representation (both activation and violation states) represent norms contents and conditions, and how that implementation can work with our domain's implementation (represented by the norms reactions to the fluents change observed). It also gives an idea of the multiple potential uses of this thesis work and the interface intuitive and simplicity of use.

7 Economic analysis

In order to analyze correctly the project's final cost, it is necessary to state all the project expenses all along the development of the same. We will split them in three: Hardware costs, software costs and human resources costs.

The hardware expenses, considering that the analysis of the problem, the development of the solution and the test of it have all been done in the same computer, are as seen in table1.

Table 1: Harware expenses

1	PC Intel Core 2 Duo T7300 Santa Rosa 2GHz	700€

Those above, are the computer characteristics on which everything has been done, but a less powerful one, and therefore cheaper, could have been used. As we said in section 3, the functionality requirements of our system could be met by not very powerful computers.

The software expenses contain all the software used during the whole developing process. These can be seen in table 2.

Table 2: Software expenses

Ubuntu 9.10	0€
OpenOffice Writer 3.1	0€
Gedit Text Editor 2.28.0	0€
Java Virtual Machine 6	0€
NetBeans IDE 6.7.1	0€
GUI Builder 1.3.2	0€
TuProlog 2.1.1	0€
SWI-Prolog 5.8	0€

As we can see, all the software used for the development of the thesis and the execution of the prototype is under license GPL, Creative Commons or equivalent, meaning the software cost of our thesis is zero.

The human resources expenses will be the main cost of this project. As we did not dispose of an expert in law, and our environmental scientist helped us for free, the costs are estimated. Those can be seen in table 3.

Table 3: Human resources expenses

Computer Scientist	Analysis + Specification + Implementation + Test	50 + 150 + 300 + 70 = 570h	30€	17.100€
Law Expert	Analysis	50h	40€	2.000€
Environmental Scientist	Analysis + Specification	50 + 50 = 100h	40€	4.000€

The total expenses of the project will be the sum of the three costs analyzed and can be seen in table 4.

Table 4: Total expenses of the project

Hardware Cost	700€
Software Cost	0€
Human Resources Cost	23.100€

8 Conclusions and future work

From the starting point of this thesis until now, our knowledge of norms and their formalization has changed dramatically. We started with almost an empty experience on the matter, studying the basic representation formulas of normative systems. Slowly we got deeper into its details, specification first and implementation at the end. By now we have achieved a significant knowledge of the whole process, the implications and the consequences of the decisions we have taken.

In this section we will see the conclusions we have reached based on the choices we have taken. We will also make reference to the aspects of this thesis where we consider some future investigation or work could be aimed at. But first, let's analyze the objectives we had for this thesis.

8.1. Project objectives

After finishing the project, an analysis of the achieved objectives must be done. Looking back at the beginning of this thesis, we have satisfied the next objectives, which were set in section 1.1:

- Obtain a generic and complete specification syntax for analyzing laws and norms.
- Use that specification to develop an implementation of real laws based on the WWTP domain.
- Develop a prototype where the norms implementation can be tested.

Once seen the covered objectives, we can focus on the conclusions.

8.2. Conclusions

8.2.1. Norm's first analysis

In section 4.1 we saw a first formalism for norms, which had been proposed and proved in some referred work. As we saw on our final specification and implementation, that formalism did not fit all of our needs in an optimal way. Some of its concepts were unused in our solution, and some were unnecessarily specific.

At this point we have concluded that the "Maintenance condition" at a specification level could be added to the "Expiration condition". At specification, only the activation and deactivation conditions bothered us, leaving apart the subtype of condition it was (maintenance condition can be understood as a subtype of activation condition), probably due to the logic formalism we chose (Situation Calculus). Later, at implementation, where each case has to be studied and implemented following a rigid syntax, those differences will appear, and it will have a much more accurate and partly different meaning (see section 5.2 "Maintenance cases").

8.2.2. Norm's definitive specification

In section 4.2.4 we proposed, discussed and applied a specification for norms. One of the main subjects of discussion in that section was how and why we treated differently two similar characteristics of norms: The activation state and the violation state. We justified those differences for investigation purposes. Now we must contemplate the results and analyze the conclusions reached.

The first element, the activation state, was treated as a dynamic element. We incorporated the concept of time-line on its definition and produced a working specification for it. The second one, violation state, was treated as a static element. We defined the domain dynamically so that its performance could be tested on it. The result is that both approaches worked out.

The dynamic representation of the activation state supposed a redundant explicitness of the temporariness element of the domain. The effect of some actions on the domain was unnecessarily stated twice, once in the definition of the actions and once in the definition of the norms. That decision, nevertheless, allowed us to prove that norm's activation state can be thought as Situation Calculus fluents, and can work the same way fluent do (see repercussions of that in next section). In an imaginary scenario where fluents did not exist, and only norms and actions define the state of the world (a totally regulated domain), we could work solely representing the time-line with norms and generate a functional solution without the use of fluents. This solution also makes norm's behavior easier to understand, as they are all self-defined in the final code.

The static representation of the violation state of norms, and the results achieved with the tests done, prove that norms can be defined based solely on static situation conditions, and that the fluents dynamic definition can deal on its own with difficulties raised by the time-line concept. That will make the development of normative systems much easier and faster. Any normative system based on a dynamic domain will only require the implementation of the domain as a dynamic element the first time, and once it has been done, we can add as many static norms definitions as we want. The changing features will already be set.

Our proposed final specification therefore depends on the goals and characteristics of the system we intend to develop. We recommend though that one choice is taken, in order to avoid doing unnecessary work.

8.2.3. Norms as fluents

In section 5.1.1.2 we saw the similarities between norms and fluents. The treatment we did of norm's "activation state" allowed us to work with both in a very similar way. As a matter of fact, we used the same implementation schema to program them. That has made us see that norms can be understood as complex fluents. Fluents with some added features.

Norms have the deontic component which rules upon its content. That deontic element had to be taken into consideration when implementing them, since the violation state depended on it. And most importantly, norms have two binary states. While fluents can only be true or false, norms can be active/inactive and violated/respected. Fact that reinforces the idea that norms are complex fluents.

The resultant implications are mostly positive, as it means that we can do with norms everything that we can do with fluents, even though they may require some more background work than simple fluents. These results prove that Situation Calculus is a very good choice when formalizing norms, as could be any formalism which uses the concept of fluent in a similar way.

8.2.4. Normative problems

As we said in the introduction of this thesis, formalizing norms which were intended for human use is no easy task. Those norms were thought with the human interpretation and reasoning capacity in mind, a resource our formalism does not have. That lack can sometimes be solved with the help of experts in the domain and the laws, as they may help obtain a generic and fixed read of the norm, but not always.

Our need of representing the content of norms in a formal and rigid way and the inherent human nature of them, clashes inevitably at times. We have realized two different norm's content may overlap, making them, and our veracious formalization, inefficient and redundant. And what is worst, occasionally we have also found contradictions between norms. That means that in a certain situation, all the possible actions that you take may lead to the violation of a norm.

In real life, that is solved by a slack interpretation of norms, but in our logic formalism, that is obviously not possible. The solution taken was making not a word by word translation of norms, but one which could help our ends the best. We must understand the norms we work with were not intended for the use we are trying to give to them.

In a future and certain scenario where computers will formally deal with normative frameworks, we must assume those norms will be done with their requirements in mind, and therefore avoid all overlapping, contradiction and free interpretation in their content. Unless, of course, a way is found to make computers deal with those problems.

8.3. Future work

8.3.1. Interface improvement

The interface developed for our prototype was done only from the developer's point of view. In order to make this prototype usable for more users and easier to handle, several modifications could be done to it. A more user friendly interface could surely be developed and further features for the exploitation of the provided information could be added to it.

8.3.2. Violation and punishment

The formal representation of norms is the first step in the process of applying them to a working system, but it is surely not the last. In order to achieve some autonomy in the resultant systems, we must add all those characteristics needed to make it work in a real life scenario. One of the most interesting ones is the punishment (or reward) related with norms, so that choices can be taken by autonomous and artificial agents who work with these formalized norms.

By our specification and implementation proposal, we know which norm has been violated in which state, and the characteristics of that violation. A very interesting addition to our system would be the definition of a punishment value assigned to every violation, which would be defined by the norm itself and the violation particularities.

The easiest way of doing it would be to define a new function which represented the amount of punishment (fine amount or other repercussions) assigned to an agent when a certain norm was violated under certain

state conditions. A deep analysis of the economic and administrative punishments established in the law would be needed, being the latest the most difficult ones to study. A unified scale of punishment would have to be defined, so that the agents could compare the repercussions of all their actions and chose the one with best results for them.

It is an interesting feature, but a difficult one. Deep legal knowledge is needed, as well as inside information of all agents, in order to know their concerns and potential set of values.

8.3.3. Multi-agent simulator

With the development of the violation state as defined in the previous section, a new possible continuation for our project becomes feasible.

Stating a goodness function on actions would allow the development of a multi-agent simulator, where a set of priorities, obligations and characteristics would be assigned to each agent. If we defined enough agents, covering all the types of our domain, we could run a simulator where the agents would interact with each other in the frame of our normative system in order to achieve its goals.

8.3.4. Legislative performance comparison

With the result of this thesis, a formalized normative system which performance can be tested, a future work which becomes possible would be to compare that performance with the one of other normative frameworks.

By the formalization of another set of laws which legislated upon the same field, we could have an easy way of comparing their performance and extract conclusions. This way we could learn the strengths and weaknesses of each one and propose modification to improve them.

In order to compare two sets of different norms we need them to use the same domain definition, that is, the same actions and fluents. It is the only way of making that comparison objective. That may not be as easy as it looks, because most legislative frameworks of the same field have a different approach to it. Once again, the knowledge of an expert in the domain would be needed.

8.4. Paper: Using Situation Calculus for Normative Agents in Urban Wastewater Systems

After considering all the work done in this thesis, we considered the conclusions reached were interesting enough to be shared with the research community. A paper was written, co-authored with Juan Carlos Nieves, Montse Aulinas and Ulises Cortes, and submitted to the 8th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS), to be held in Salamanca, the 26-28th of April of 2010.

The paper was accepted and will be published in the *Advances in Intelligent and Soft-Computing* series of Springer [30]. We have included the paper in Annex VI.

9 References

[1] Alain Colmerauer & Phillipe Roussel. *The birth of Prolog*. History of programming languages, pag. 331-367, 1996.

[2] Albert J.J. Anglberger. *Dynamic Deontic Logic and its Paradoxes*. Studia Logica, pag. 427-435, 2008.

[3] Andy Salter & Kecheng Liu. *Using semantic analysis and norm analysis to model organizations*. International Symposium on Computer Science and Computational Technology, 2008.

[4] A.B. Baker. *Nonmonotonic Reasoning in the Framework of the Situation Calculus*. Artificial Intelligence, vol 49, pag. 5, 1991.

[5] CEC. *Council Directive 91/271/EEC of 21 May 1991 concerning urban wastewater treatment*. Official Journal, 135, 1991.

[6] Davide Grossi & Frank Dignum. *From abstract to concrete norms in institutions*. Lecture Notes in Computer Science, pag. 12-29, 2005.

[7] Diari Oficial de la Generalitat de Catalunya. *Decret 130/2003*. Núm. 3894-29.5.2003, pag. 11143-11158.

[8] Frank Dignum. *Autonomous agents with norms*. Artificial Intelligence and Law, pag. 69-79, 1997.

[9] *Gedit*. URL: http://projects.gnome.org/gedit/.

[10] G.H. von Wright. *Deontic logic*. Mind, 1951.

[11] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin & Richard B. Scherl. *GOLOG: A Logic Progamming Language for Dynamic Domains*. Journal of Logic Programming, vol. 31, 1997.

[12] Huib Aldewereld & Virginia Dignum. *Operetta syntax landmarks and norms*.

[13] *Java*. URL: http://java.sun.com/.

[14] John Bryant. *The Logic of Relative Modality and the Paradoxes of Deontic Logic*. Notre Dame J. Formal Logic, Vol. 21, pag. 78-88, 1980.

[15] John McCarthy & Patrick Hayes. *Some philosophical problems from the standpoint of artificial intelligence*. Machine Intelligence 4, 1969.

[16] John W. Lloyd. *Foundations of Logic Programming*, 1987.

[17] Juan Carlos Nieves, Montse Aulinas & Ulises Cortés. *Reasoning About Actions for the Management of Urban Wastewater Systems*. Frontiers in Artificial Intelligence and Applications, Vol. 202, pag. 371-380, 2009.

[18] J.-J. Ch. Meyer. *A Different Approach to Deontic Logic: Deontic Logic Viewed as a Variant of Dynamic Logic*. Notre Dame J. Formal Logic, Vol. 29, pag. 109-136, 1987.

[19] J.-J. Ch. Meyer, F.P.M. Dignum & R.J. Wieringa. *A Model for Organizational Interaction: Based on Agents Founded in Logic*, 2003.

[20] J.-J. Ch. Meyer, F.P.M. Dignum & R.J. Wieringa. *The Paradoxes of Deontic Logic Revisited: A Computer Science Perspective*. Technical Report UU-CS-1994-38, Department of Information and Computing Sciences, Utrecht University, 1994.

[21] J.-J. Ch. Meyer & R.J. Wieringa. *Deontic Logic in Computer Science: Normative System Specification*, pp 1-36, 1991.

[22] Michael Thielscher. *The Concurrent, Continuous Fluent Calculus*. Studia Logica, pag. 315-331, 2004.

[23] Montse Aulinas. *Management of Industrial Wastewater Discharges in River Basins Through Agent's Argumentation*, PhD Thesis, Universitat de Girona, 2009.

[24] M. Sànchez-Marrè, K.Gilbert, R. Sojda, J. Steyer, P. Struss, I. Rodríguez-Roda, J. Comas, V. Brilhante & E. Roehl. *Intelligent Environmental Decision Support Systems*, 2009.

[25] *NetBeans*. URL: http://netbeans.org/.

[26] Nir Oren, Sofia Panagiotidi, Javier Vázquez-Salceda, Sanjay Modgil, Michael Luck & Simon Miles. *Towards a Formalisation of Electronic Contracting Environments*. Proceedings of Coordination, Organization, Institutions and Norms in Agent Systems, the International Workshop at AAAI, 2008

[27] Rob Miller. *Situation Calculus Specifications for Event Calculus Logic Programs*. Proceedings of the Third International Conference on Logic Programming and Non-monotonic Reasoning, 1995.

[28] Robert Kowalski & Marek Sergot. *A logic based calculus of events*. New Generation Computing archive. Vol. 4, 1986.

[29] R. Reiter. *The frame problem in Situation Calculus: a simple solution (sometimes) and a completeness result for goal regression*. Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy, pag. 359 – 380, 1991.

[30] *Springer*. URL: http://www.springer.com/.

[31] Stanford Encyclopedia of Philosophy. *Deontic Logic*, 2006.

[32] *SWI-Prolog*. URL: http://www.swi-prolog.org/.

[33] *Tuprolog*. URL: http://alice.unibo.it/xwiki/bin/view/Tuprolog/.

[34] Wamberto W. Vasconcelos. *Norm verification and analysis of electronic institutions*. Vol. 3476 of LNAI, 2004.

10 Annex I: The Catalan Sanitation Plan studied norms

Norm's analyzed, specified, implemented and tested in this thesis

7.1

For the next agents is obligatory to obtain an authorization and to respect the restrictions of Annex[1] I and II:

- Non domestic users whose activity is included in C, D and E sections of Economic Activities Catalan Classification (Decree 97/1995) considered potential pollutant agents.

- Those who generate spills > 6.000 m3/year.

7.2.

If the agent is a domestic or similar user then Annex I

7.3.

Only if the pertinent agent consider it is best to spill to the environment then it is possible to do not spill to the sewage system.

7.4.

If new spill then the pertinent agent will register it in a census (article 18)

8.1.

Prohibitions:

a) Substances of Annex I

[1] The mentioned Annex I and II refer to the Catalan Sanitation Plan Annex I and II. The first one contains a list of the limited substances and their limitations, and the second one contains a list of the forbidden substances.

b) To dilute in order to approach best levels; if there is an emergency or an imminent risk then it is possible to dilute with previous warning to the competent agent.

c) To spill white waters to the public sewer system; if there is not an alternative to spill them (separative net or a river) then one must obtain an authorization to do these spills.

8.2.

If domestic spills contain substances included in Annex II then they must respect the established limitations.

10.2.

If you have the authorization then you can spill black waters to the public sewer system according to the regulations established.

11 Annex II: WWTP domain's actions and fluents

Decomposition of the WWTP domain in Situation Calculus' actions and fluents syntax

ACTIONS

- add_substance(IdSpill,Substance,Quantity)
- communicate_dilution(IdAgent,IdAgentEntity,IdSpill,Size)
- del_substance(IdSpill,Substance,Quantity)
- del_total_substance(IdSpill,Substance)
- make_spill(IdSpill,IdAgent)
- register_spill(IdSpill,census, IdAgentEntity)
- set_agent_activity(IdAgent,Activity)
- set_agent_situation(IdAgent,Situation)
- set_agent_type(IdAgent,AgentType)
- set_associated_entity(IdAgent,IdAgentEntity)
- set_best_place_to_spill(IdSpill,SpillPlace,IdAgent)
- set_pollutant_activity(Activity)
- set_spill_authorization(IdSpill,SpillPlace,IdAgent,IdAgentEntity,Authorization)
- set_spill_not_possible(IdAgent,IdSpill,SpillPlace)
- set_spill_possible(IdAgent,IdSpill,SpillPlace)
- create_spill(IdSpill)
- cancel_spill(IdSpill,IdAgent)
- unset_pollutant_activity(Activity)
- set_substance_limitation(Substance,Limit)
- set_spill_place(IdSpill,SpillPlace)

- set_spill_type(IdSpill,SpillType)
- new_agent(IdAgent)
- delete_agent(IdAgent)
- dilute_spill(IdSpill,Dilution)

FLUENTS

- agent_activity(IdAgent,Activity)
- agent_associated_entity(IdAgent,IdAgentEntity)
- agent_situation(IdAgent,risk/emergency)
- agent_type(IdAgent,AgentType)
- agent_spill(IdAgent,IdSpill)
- agent_spills_total_size(IdAgent,Size)
- best_place_to_spill(IdSpill,Place,IdAgentEntity)
- dilution_comunicated(IdAgent,IdAgentEntity,IdSpill,Size)
- pollutant_activity(Activity)
- pollutant_agent(IdAgent)
- spill_authorized(IdSpill,sewer_system,IdAgent,IdAgentEntity)
- substance_limitation(Substance,Limit)
- spill_contains_limited_substance(IdSpill,Substance)
- spill_not_possible(IdAgent,IdSpill,SpillPlace)
- spill_type(IdSpill,black_waters)
- substance_spill_quantity(IdSpill,Substance,Quantity)
- spill_total_size(IdSpill,SizeS)
- spill_registered(IdSpill,RegisterPlace, IdAgentEntity)
- agent_limited_substance(IdAgent,Substance,Quantity)
- agent_substance(IdAgent,Substance,Quantity)
- spill_possible(IdAgent,IdSpill,SpillPlace)
- spill_respects_limitations(IdSpill)

- spill_violates_limitation(IdSpill,Substance)
- spill_place(IdAgent,SpillPlace)
- agent(IdAgent)
- spill(IdSpill)
- spill_diluted(IdSpill,Dilution)

12 Annex III: First approach to specification of norms

Specification of the selected norms following the formalism seen in section 4.1

Norm 7.1

Id:

71

ActivCond:

agent_spills_per_year > 6000 V

(agent_type_is_non_domestic Λ agent_is_potential_pollutant)

MaintCond:

agent_spills_per_year > 6000 V agent_is_potential_pollutant

ExpirCond:

agent_spills_per_year < 6000 Λ agent_is_not_potential_pollutant

Op:

O (Obligatory)

RoleorActor:

spill_producer_agent

NormContent:

∀Spill Sp: Sp_done_by_agent → (S_authorized_by_associated_entity Λ

∀Substance Sub: Sp_contains_Sub → (Sub_respects_limitation Λ Sub_not_forbidden))

Norm 7.2

Id:

72

ActivCond:

agent_type_is_domestic

MaintCond:

True

(The norm cannot be deactivated once it is active)

ExpirCond:

False

(The norm cannot be deactivated once it is active)

Op:

F (Forbidden)

RoleorActor:

spill_producer_agent

NormContent:

$\forall$*Spill Sp: Sp_done_by_agent* → ($\forall$ *Substance Sub: Sp_contains_Sub* →

(Sub_not_forbidden))

Norm 7.3

Id:

731

ActivCond:

True

(The norm is always active)

MaintCond:

not_best_place_to_spill_is_environment

ExpirCond:

False

(The norm is always active)

Op:

F (Forbidden)

RoleorActor:

spill_producer_agent

NormContent:

∃ Spill Sp: Sp_done_by_agent → (Sp_spilled_at_environment)

Id:

732

ActivCond:

(agent_type_is_WLA ∨ agent_type_is_CWA) ∧ agent_best_place_to_spill_is_environment

MaintCond:

best_place_to_spill_is_environment

ExpirCond:

best_place_to_spill_is_not_environment

Op:

P (Possible)

RoleorActor:

spill_producer_agent

NormContent:

∃ Spill Sp: Sp_done_by_agent → (Sp_spilled_at_environment)

Norm 7.4

Id:

74

ActivCond:

new_spill_done

MaintCond:

True

(The norm cannot be deactivated once it is active)

ExpirCond:

False

(The norm cannot be deactivated once it is active)

Op:

O (Obligatory)

RoleorActor:

spill_producer_agent

NormContent:

∃ Agent Ag: Ag_is_associated_entity_to_agent → (Ag_registered_spill_in_census)

Norm 8.1 a)

Id:

81

ActivCond:

True

(The norm is always active)

MaintCond:

True

(The norm is always active)

ExpirCond:

False

(The norm is always active)

Op:

F (Forbidden)

RoleorActor:

spill_producer_agent

NormContent:

∃ Spill Sp: Sp_done_by_agent → (Δ Substance Sub: Sp_contains_Sub → (Sub_forbidden))

Norm 8.1 b)

Id:

8121

ActivCond:

True

(The norm is always active)

MaintCond:

(agent_not_in_risk_situation Λ agent_not_in_emergency_situation) V pertinent_agent_not_warned

ExpirCond:

False

(The norm is always active)

Op:

F (Forbidden)

RoleorActor:

spill_producer_agent

NormContent:

∃ Spill Sp: Sp_done_by_agent → Sp_diluted

Id:

8122

ActivCond:

(agent_in_risk_situation V agent_in_emergency_situation) Λ pertinent_agent_warned

MaintCond:

agent_in_risk_situation V agent_in_emergency_situation

ExpirCond:

agent_not_in_risk_situation Λ agent_not_in_emergency_situation

Op:

P (Possible)

RoleorActor:

spill_producer_agent

NormContent:

∃ Spill Sp: Sp_done_by_agent → Sp_diluted

Norm 8.1 c)

Id:

8131

ActivCond:

True

(The norm is always active)

MaintCond:

spill_not_white_waters V spill_possible_in_separative_net V spill_possible_in_river V spill_not_authorizated

ExpirCond:

False

(The norm is always active)

Op:

F (Forbidden)

RoleorActor:

spill_producer_agent

NormContent:

∃ Spill Sp: Sp_done_by_agent → Sp_in_sewer_system

Id:

8132

ActivCond:

Spill_white_waters ∧ spill_not_possible_in_separative_net ∧ spill_not_possible_in_river ∧ spill_authorizated

MaintCond:

spill_not_possible_in_separative_net ∧ spill_not_possible_in_river

ExpirCond:

spill_possible_in_separative_net ∨ spill_possible_in_river

Op:

P (Possible)

RoleorActor:

spill_producer_agent

NormContent:

∃ Spill Sp: Sp_done_by_agent → Sp_in_sewer_system

Norm 8.2

Id:

82

ActivCond:

agent_type_is_domestic ∧ spill_contains_limited_substances

MaintCond:

spill_contains_limited_substances

ExpirCond:

spill_not_contains_limited_substances

Op:

F (Forbidden)

RoleorActor:

spill_producer_agent

NormContent:

∃ Spill Sp: Sp_done_by_agent → (∃ Substance Sub: Sp_contains_Sub →

(Sub_quantity_bigger_than_limitation))

Norm 10.2

Id:

1021

ActivCond:

True

(The norm is always active)

MaintCond:

spill_not_authorizated ∨ spill_violates_limitations

ExpirCond:

False

(The norm is always active)

Op:

F (Forbidden)

RoleorActor:

spill_producer_agent

NormContent:

∃ Spill Sp: Sp_done_by_agent → (Sp_is_black_waters ∧ Sp_in_sewer_system)

Id:

1022

ActivCond:

spill_authorizated $\wedge$ *spill_not_violates_limitations*

MaintCond:

spill_not_violates_limitations

ExpirCond:

spill_violates_limitations

Op:

P (Possible)

RoleorActor:

spill_producer_agent

NormContent:

$\exists$ *Spill Sp: Sp_done_by_agent* $\rightarrow$ *(Sp_is_black_waters* $\wedge$ *Sp_in_sewer_system)*

13 Annex IV: Final specification

Specification of the selected norms following the formalism proposed in section 4.2

Norm 71

Operator:

O (Obligatory)

State conditions for activation:

(domestic_agent ∧ potential_pollutant_agent) ∨ m3_generated_by_agent > 6000

Activation actions:

- *make a spill.*

  Activates norm if: With that spill the agent generates more than 6000 m3/year.

- *add a substance to a spill.*

  Activates norm if: With the added quantity the agent generates more than 6000 m3/year.

- *set the type of the agent to non domestic.*

  Activates norm if: The agent was potential pollutant.

- *set the agent activity to one considered pollutant.*

  Activates norm if: The agent was domestic.

- *set an activity to be considered pollutant.*

  Activates norm if: The agent was domestic and it was doing that activity.

Termination actions:

- *set the type of the agent to non "non domestic".*

  Terminates norm if: The agent generates spills < 6000 m3/year.

- *set the agent activity to one not considered pollutant.*

  Terminates norm if: The agent generates spills < 6000 m3/year.

- *set the agents current activity to non pollutant.*

  Terminates norm if: The agent generates spills < 6000 m3/year.

- *cancel a spill.*

  Terminates norm if: The agent spills quantity becomes less than 6000 and the agent is not "non domestic" or is not a pollutant agent.

- *delete a substance from a spill.*

  Terminates norm if: The agent spills quantity becomes less than 6000 and the agent is not "non domestic" or is not a pollutant agent.

Norm content:

Obtain an authorization and respect the restrictions of Annex I and II.

Norm 7.2

Operator:

F (Forbidden)

State conditions for activation:

domestic_agent

Activation actions:

- *set the type of the agent as domestic.*

  Activates norm if: Always.

Termination actions:

- *set the type of the agent to non "non domestic".*

  Terminates norm if: Always.

Norm content:

Spill forbidden substances.

Norm 731

Operator:

F (Forbidden)

State conditions for activation:

Norm 732 not active

Norm content:

Spill in the sewage system.

Norm 732

Operator:

P (Possible)

State conditions for activation:

The pertinent agent considers best to spill in the environment.

Activation actions:

- *An agent changes the place it considers best to spill.*

  Activates norm if: The agent is the pertinent agent and the place is the environment.

- *The agent changes its pertinent agent.*

  Activates norm if: The new pertinent agent considers best place the environment.

Termination actions:

- *An agent changes the place it considers best to spill.*

Terminates norm if: The agent is the pertinent agent and the new place is not the environment.

- *The agent changes its pertinent agent.*

  Terminates norm if: The new pertinent agent do not consider best to spill in the environment.

Norm content:

Spill in the sewage system.

Norm 74

Operator:

O (Obligatory)

State conditions for activation:

A new, unregistered spill has been done.

Activation actions:

- *make a spill.*

  Activates norm if: The new spill is not registered by the pertinent agent.

Termination actions:

Cannot be deactivated.

Norm content:

The spill is registered in the census by the pertinent agent.

Norm 811

Operator:

F (Forbidden)

State conditions for activation:

Always active.

Activation actions:

Always active.

Termination actions:

Always active.

Norm content:

Spill forbidden substances.

Norm 8121

Operator:

F (Forbidden)

State conditions for activation:

Norm 8122 not active.

Norm content:

Dilute a spill.

Norm 8122

Operator:

P (Possible)

State conditions for activation:

There is an emergency or risk situation and the pertinent agent has been warned.

Activation actions:

- *The situation changes.*

  Activates norm if: The new situation is emergency or risk and the pertinent agent was already warned.

- *Warn an agent.*

  Activates norm if: The situation is emergency or risk and the warned agent is the pertinent agent.

- *Change the pertinent agent.*

  Activates norm if: The situation is emergency or risk and the new pertinent agent was already warned.

Termination actions:

- *Change the situation.*

  Terminates norm if: The new situation is not emergency nor risk.

Norm content:

Dilute a spill.

Norm 8131

Operator:

F (Forbidden)

State conditions for activation:

Norm 8132 not active

Norm content:

Spill white waters to the public sewer system.

Norm 8132

Operator:

P (Possible)

State conditions for activation:

There is no alternative to spill them and it is authorized by the pertinent agent.

Activation actions:

- *A spilling place is set as impossible.*

  Activates norm if: The only left possible place to spill is the sewer system and the spill is authorized by the pertinent agent.

- *A spill authorization is received.*

  Activates norm if: The only possible place to spill is the sewer system and the authorizer agent is the pertinent agent.

- *Change the pertinent agent.*

  Activates norm if: The new agent has authorized the spill and the only possible place to spill is the sewer system.

Termination actions:

- *Set a spill place as possible.*

  Terminates norm if: The spill place is not the sewer system.

- *Set a spill authorization.*

  Terminates norm if: The authorization is negative and is done by the pertinent agent.

Norm content:

Spill white waters to the public sewer system.

Norm 82

Operator:

F (Forbidden)

State conditions for activation:

The spill contains limited substances and the agent is domestic.

Activation actions:

- *Change the agent's type.*

  Activates norm if: The new agent type is domestic and the spill contained limited substances.

- *Add a substance to the spill.*

  Activates norm if: The added substance is limited and the agent is domestic.

Termination actions:

- *Delete a substance to the spill.*

  Terminates norm if: The substance was the last limited substance of the spill.

- *Change the agent's type.*

  Terminates norm if: The new agent's type is not domestic.

Norm content:

The spill respects the established limitations.

Norm 1021

Operator:

F (Forbidden)

State conditions for activation:

Norm 1022 not active.

Norm content:

Spill black waters to the public sewer system.

Norm 1022

Operator:

P (Possible)

State conditions for activation:

The spill is authorized and it respects the substance limitations.

Activation actions:

- *Get a spill authorization.*

  Activates norm if: The authorization is positive and done by the pertinent agent and the spill respects the established limitations.

- *Change the pertinent agent.*

  Activates norm if: The new pertinent agent has authorized the spill and it respects the established limitations.

- *Delete substance from the spill*

  Activates norm if: The spill is authorized by the pertinent agent, and by deleting the substance, all the established limitations are respected.

Termination actions:

- *Get a spill authorization.*

  Terminates norm if: The authorization is negative and done by the pertinent agent.

- *Add substance to the spill.*

  Terminates norm if: By adding the substance, an established limitation is violated.

Norm content:

Spill black waters to the public sewer system.

14 Annex V: Domain and norms implementation

Prolog code representing the Situation Calculus elements and the rest of the required fluents and actions of the WWTP domain.

```
%%%NORMATIVE VIOLATIONS

%%%%%%%%%violated(norm(71,IdAgent),S)
violated(norm(71,IdAgent),S) :-

holds(norm(71,IdAgent),S), holds(agent_spill(IdAgent,IdSpill),S),
holds(spill_violates_limitation(IdSpill,Substance),S);

holds(norm(71,IdAgent),S), holds(agent_spill(IdAgent,IdSpill),S),
holds(spill_place(IdAgent,SpillPlace),S),
holds(agent_associated_entity(IdAgent,IdAgentEntity),S),
\+holds(spill_authorized(IdSpill,SpillPlace,IdAgent,IdAgentEntity),S).

%%%%%%%%%violated(norm(72,IdAgent),S)
violated(norm(72,IdAgent),S) :-

holds(norm(72,IdAgent),S), holds(agent_spill(IdAgent,IdSpill),S),
holds(spill_violates_limitation(IdSpill,Substance),S).


%%%%%%%%%violated(norm(731,IdAgent,IdSpill),S)
violated(norm(731,IdAgent,IdSpill),S) :-

holds(norm(731,IdAgent,IdSpill),S),
holds(spill_place(IdSpill,SpillPlace),S),SpillPlace\=sewer_system.

%%%%%%%%%violated(norm(74,IdAgent,IdSpill),S)
violated(norm(74,IdAgent,IdSpill),S) :-

holds(norm(74,IdAgent,IdSpill),S),
holds(agent_associated_entity(IdAgent,IdAgentEntity),S),
\+holds(spill_registered(IdSpill,census,IdAgentEntity),S).

%%%%%%%%%violated(norm(81,IdAgent),S)
violated(norm(81,IdAgent,Substance),S) :-

holds(norm(81,IdAgent),S), holds(agent_spill(IdAgent,IdSpill),S),
holds(spill_violates_limitation(IdSpill,Substance),S),
holds(substance_limitation(Substance,0),S).

%%%%%%%%%violated(norm(8121,IdAgent,IdSpill),S)
violated(norm(8121,IdAgent,IdSpill),S) :-
```

```
holds(norm(8121,IdAgent,IdSpill),S), holds(spill_diluted(IdSpill,Q),S),Q>0.

%%%%%%%%%violated(norm(8131,IdAgent,IdSpill),S)
violated(norm(8131,IdAgent,IdSpill),S) :-

holds(norm(8131,IdAgent,IdSpill),S), holds(spill_type(IdSpill,white_waters),S),
holds(spill_place(IdAgent,sewer_system),S).

%%%%%%%%%violated(norm(82,IdAgent,IdSpill),S)
violated(norm(82,IdAgent,IdSpill),S) :-

holds(norm(82,IdAgent,IdSpill),S),
holds(spill_violates_limitation(IdSpill,Substance),S).

%%%%%%%%%violated(norm(1021,IdAgent,IdSpill),S)
violated(norm(1021,IdAgent,IdSpill),S) :-

holds(norm(1021,IdAgent,IdSpill),S), holds(spill_type(IdSpill,black_waters),S),
holds(spill_place(IdSpill,sewer_system),S).

%%%POSSIBLE ACTIONS

poss(new_agent(IdAgent),S):-
\+holds(agent(IdAgent),S).

poss(create_spill(IdSpill),S):-
\+holds(spill(IdSpill),S).

poss(set_agent_situation(IdAgent,Sit),S):-
holds(agent(IdAgent),S), situation(Sit).

poss(set_agent_activity(IdAgent,Act),S):-
holds(agent(IdAgent),S), activity(Act).

poss(set_agent_type(IdAgent,AgentType),S):-
holds(agent(IdAgent),S), agentType(AgentType).

poss(make_spill(IdSpill,IdAgent),S):-
holds(agent(IdAgent),S), holds(spill(IdSpill),S),
holds(agent_type(IdAgent,domestic),S);
holds(agent(IdAgent),S), holds(spill(IdSpill),S),
holds(agent_type(IdAgent,non_domestic),S).

poss(set_associated_entity(IdAgent,IdAgentEntity),S):-
holds(agent(IdAgent),S), holds(entity_agent(IdAgentEntity),S).

poss(add_substance(IdSpill,Substance,Quantity),S):-
holds(spill(IdSpill),S), substance(Substance), Quantity>=0.

poss(communicate_dilution(IdAgent,IdAgentEntity,IdSpill,Size),S):-
holds(agent(IdAgent),S), holds(entity_agent(IdAgentEntity),S),
holds(spill(IdSpill),S), holds(agent_spill(IdAgent,IdSpill),S), Size>0.

poss(del_substance(IdSpill,Substance,Quantity),S):-
```

```
holds(spill(IdSpill),S), substance(Substance),
holds(substance_spill_quantity(IdSpill,Substance,QPre),S), Quantity>=0,
Quantity=<QPre.

poss(del_total_substance(IdSpill,Substance),S):-
holds(spill(IdSpill),S), substance(Substance).

poss(register_spill(IdSpill,census, IdAgentEntity),S):-
holds(spill(IdSpill),S), holds(entity_agent(IdAgentEntity),S).

poss(set_best_place_to_spill(IdSpill,SpillPlace,IdAgent),S):-
holds(entity_agent(IdAgent),S), holds(spill(IdSpill),S), spillPlace(SpillPlace).

poss(set_pollutant_activity(Activity),S):-
activity(Activity).

poss(set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,Authorization,SpillPla
ce),S):-
holds(agent(IdAgent),S), holds(entity_agent(IdAgentEntity),S),
holds(spill(IdSpill),S), spillPlace(SpillPlace), authorization(Authorization).

poss(set_spill_not_possible(IdAgent,IdSpill,SpillPlace),S):-
holds(agent(IdAgent),S), holds(spill(IdSpill),S), spillPlace(SpillPlace).

poss(set_spill_possible(IdAgent,IdSpill,SpillPlace),S):-
holds(agent(IdAgent),S), holds(spill(IdSpill),S), spillPlace(SpillPlace).

poss(cancel_spill(IdSpill,IdAgent),S):-
holds(agent(IdAgent),S), holds(spill(IdSpill),S),
holds(agent_spill(IdAgent,IdSpill),S).

poss(delete_agent(IdAgent),S):-
holds(agent(IdAgent),S), \+holds(agent_spill(IdAgent,IdSpill),S).

poss(unset_pollutant_activity(Activity),S):-
activity(Activity).

poss(set_substance_limitation(Substance,Limit),S):-
substance(Substance).

poss(set_spill_place(IdSpill,SpillPlace),S):-
holds(spill(IdSpill),S), spillPlace(SpillPlace).

poss(set_spill_type(IdSpill,SpillType),S):-
holds(spill(IdSpill),S), spillType(SpillType).

poss(dilute_spill(IdSpill,Dilution),S):-
holds(spill(IdSpill),S).

%%%END POSSIBLE ACTIONS

%%%RULES

%%%%%%%%%agent(IdAgent)
holds(agent(IdAgent),do(A,S)) :-
A = new_agent(IdAgent), poss(A,S);
```

```
holds(agent(IdAgent),S), \+ A = delete_agent(IdAgent), poss(A,S).

%%%%%%%%%spill(IdSpill)
holds(spill(IdSpill),do(A,S)) :-
A = create_spill(IdSpill), poss(A,S);
holds(spill(IdSpill),S), \+ A = cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%entity_agent(IdAgent)
holds(entity_agent(IdAgent),do(A,S)) :-
A = set_agent_type(IdAgent,wla), poss(A,S);
A = set_agent_type(IdAgent,wca), poss(A,S);
holds(entity_agent(IdAgent),S), \+ A = set_agent_type(IdAgent,Type), poss(A,S).

%%%%%%%%%agent_activity(IdAgent,Activity)
holds(agent_activity(IdAgent,Activity),do(A,S)) :-
A = set_agent_activity(IdAgent,Activity), poss(A,S);
holds(agent_activity(IdAgent,Activity),S),
\+A=set_agent_activity(IdAgent,Activity2), \+A=delete_agent(IdAgent), poss(A,S).

%%%%%%%%%agent_associated_entity(IdAgent,IdAgentEntity)
holds(agent_associated_entity(IdAgent,IdAgentEntity),do(A,S)) :-
A = set_associated_entity(IdAgent,IdAgentEntity), poss(A,S);
holds(agent_associated_entity(IdAgent,IdAgentEntity),S),
\+A=set_associated_entity(IdAgent,IdAgentEntity2), \+A=delete_agent(IdAgent),
\+A=delete_agent(IdAgentEntity), poss(A,S).

%%%%%%%%%agent_situation(IdAgent,Situation)
holds(agent_situation(IdAgent,Sit),do(A,S)) :-
A = set_agent_situation(IdAgent,Sit), poss(A,S);
holds(agent_situation(IdAgent,Sit),S), \+A=set_agent_situation(IdAgent,Sit2),
\+A=delete_agent(IdAgent), poss(A,S).

%%%%%%%%%agent_type(IdAgent,AgentType)
holds(agent_type(IdAgent,AgentType),do(A,S)) :-
A = set_agent_type(IdAgent,AgentType), poss(A,S);
holds(agent_type(IdAgent,AgentType),S), \+A=set_agent_type(IdAgent,AgentType2),
\+A=delete_agent(IdAgent), poss(A,S).

%Inicialització per defecte dels agents a no domestics
holds(agent_type(IdAgent,non_domestic),do(A,S)) :-
A = new_agent(IdAgent), poss(A,S).

%%%%%%%%%agent_spill(IdAgent,IdSpill)
holds(agent_spill(IdAgent,IdSpill),do(A,S)) :-
A=make_spill(IdSpill,IdAgent), poss(A,S);
holds(agent_spill(IdAgent,IdSpill),S) ,\+A=cancel_spill(IdSpill,IdAgent),
\+A=delete_agent(IdAgent), poss(A,S).

%%%%%%%%%agent_limited_substance(IdAgent,Substance,Quantity)
holds(agent_limited_substance(IdAgent,Substance,Quantity),do(A,S)) :-
A = make_spill(IdSpill,IdAgent),
holds(substance_spill_quantity(IdSpill,Substance,Quantity),S), Quantity>0,
holds(substance_limitation(Substance,Limit),S),
\+holds(agent_limited_substance(IdAgent,Substance,Q),S), poss(A,S);
```

```
A = add_substance(IdSpill,Substance,QAdded),
holds(agent_spill(IdAgent,IdSpill),S),
holds(agent_limited_substance(IdAgent,Substance,QIni),S), Quantity is
QIni+QAdded, poss(A,S);

A = add_substance(IdSpill,Substance,Quantity),
holds(agent_spill(IdAgent,IdSpill),S),
holds(substance_limitation(Substance,Limit),S),
\+holds(agent_limited_substance(IdAgent,Substance,Q),S), poss(A,S);

A = del_substance(IdSpill,Substance,QDel), holds(agent_spill(IdAgent,IdSpill),S),
holds(agent_limited_substance(IdAgent,Substance,QPre),S), Quantity is QPre-QDel,
poss(A,S);

A = del_total_substance(IdSpill,Substance),
holds(agent_spill(IdAgent,IdSpill),S),
holds(substance_limitation(Substance,Limit),S),
holds(substance_spill_quantity(IdSpill,Substance,QDel),S),
holds(agent_limited_substance(IdAgent,Substance,QPre),S), Quantity is QPre-QDel,
poss(A,S);

A = set_substance_limitation(Substance,Limit),
holds(agent_substance(IdAgent,Substance,Quantity),S), poss(A,S);

holds(agent_limited_substance(IdAgent,Substance,Quantity2),S), A =
make_spill(IdSpill,IdAgent),
holds(substance_spill_quantity(IdSpill,Substance,QAdded),S), Quantity is
Quantity2+QAdded, poss(A,S);

holds(agent_limited_substance(IdAgent,Substance,Quantity2),S), A =
cancel_spill(IdSpill,IdAgent),
holds(substance_spill_quantity(IdSpill,Substance,QDel),S), Quantity is Quantity2-
QDel, poss(A,S);

holds(agent_limited_substance(IdAgent,Substance,Quantity),S), A =
add_substance(IdSpill,Substance,QAdded), \+holds(agent_spill(IdAgent,IdSpill),S),
poss(A,S);

holds(agent_limited_substance(IdAgent,Substance,Quantity),S), A =
add_substance(IdSpill,Substance,0), poss(A,S);

holds(agent_limited_substance(IdAgent,Substance,Quantity),S), A =
del_substance(IdSpill,Substance,QDel), \+holds(agent_spill(IdAgent,IdSpill),S),
poss(A,S);

holds(agent_limited_substance(IdAgent,Substance,Quantity),S), A =
del_substance(IdSpill,Substance,0), poss(A,S);

holds(agent_limited_substance(IdAgent,Substance,Quantity),S), A =
del_total_substance(IdSpill,Substance), \+holds(agent_spill(IdAgent,IdSpill),S),
poss(A,S);

holds(agent_limited_substance(IdAgent,Substance,Quantity),S), A =
set_substance_limitation(Substance,Limit), Limit\=0, poss(A,S);
```

```
holds(agent_limited_substance(IdAgent,Substance,Quantity),S),
\+A=make_spill(IdSpill,IdAgent), \+A=add_substance(IdSpill,Substance,QAdded),
\+A=del_substance(IdSpill,Substance,QDel),
\+A=del_total_substance(IdSpill,Substance),
\+A=set_substance_limitation(Substance,Limit), \+A=cancel_spill(IdSpill,IdAgent),
\+A=delete_agent(IdAgent), poss(A,S).

%%%%%%%%%agent_substance(IdAgent,Substance,Quantity)
holds(agent_substance(IdAgent,Substance,Quantity),do(A,S)) :-
A = make_spill(IdSpill,IdAgent),
holds(agent_substance(IdAgent,Substance,QIni),S),
holds(substance_spill_quantity(IdSpill,Substance,QAdded),S), Quantity is
QIni+QAdded, poss(A,S);

A = make_spill(IdSpill,IdAgent),
holds(substance_spill_quantity(IdSpill,Substance,Quantity),S), Quantity>0,
\+holds(substance_limitation(Substance,Limit),S),
\+holds(agent_substance(IdAgent,Substance,Q),S), poss(A,S);

A = add_substance(IdSpill,Substance,QAdded),
holds(agent_spill(IdAgent,IdSpill),S),
holds(agent_substance(IdAgent,Substance,QIni),S), Quantity is QIni+QAdded,
poss(A,S);

A = add_substance(IdSpill,Substance,Quantity), Quantity>0,
holds(agent_spill(IdAgent,IdSpill),S),
\+holds(substance_limitation(Substance,Limit),S),
\+holds(agent_substance(IdAgent,Substance,Q),S), poss(A,S);

%A = add_substance(IdSpill,Substance,Quantity), Quantity>0,
holds(agent_spill(IdAgent,IdSpill),S),
\+holds(substance_limitation(Substance,Limit),S),
\+holds(agent_substance(IdAgent,Substance,Q),S), poss(A,S);

A = del_substance(IdSpill,Substance,QDel), holds(agent_spill(IdAgent,IdSpill),S),
holds(agent_substance(IdAgent,Substance,QPre),S), Quantity is QPre-QDel,
poss(A,S);

A = del_total_substance(IdSpill,Substance),
holds(agent_spill(IdAgent,IdSpill),S),
holds(substance_spill_quantity(IdSpill,Substance,QDel),S),
holds(agent_substance(IdAgent,Substance,QPre),S), Quantity is QPre-QDel,
poss(A,S);

A = cancel_spill(IdSpill,IdAgent),
holds(substance_spill_quantity(IdSpill,Substance,QDel),S),
holds(agent_substance(IdAgent,Substance,QPre),S), Quantity is QPre-QDel,
poss(A,S);

holds(agent_substance(IdAgent,Substance,Quantity),S), A =
add_substance(IdSpill,Substance,QAdded), \+holds(agent_spill(IdAgent,IdSpill),S),
poss(A,S);

holds(agent_substance(IdAgent,Substance,Quantity),S), A =
add_substance(IdSpill,Substance,0), poss(A,S);
```

```
holds(agent_substance(IdAgent,Substance,Quantity),S), A =
del_substance(IdSpill,Substance,QDel), \+holds(agent_spill(IdAgent,IdSpill),S),
poss(A,S);

holds(agent_substance(IdAgent,Substance,Quantity),S), A =
del_substance(IdSpill,Substance,0), poss(A,S);

holds(agent_substance(IdAgent,Substance,Quantity),S), A =
del_total_substance(IdSpill,Substance), \+holds(agent_spill(IdAgent,IdSpill),S),
poss(A,S);

%holds(agent_substance(IdAgent,Substance,Quantity),S), A =
cancel_spill(IdSpill,IdAgent), holds(spill_total_size(IdSpill,0),S), poss(A,S);


holds(agent_substance(IdAgent,Substance,Quantity),S),
\+A=make_spill(IdSpill,IdAgent), \+A=add_substance(IdSpill,Substance,QAdded),
\+A=del_substance(IdSpill,Substance,QDel),
\+A=del_total_substance(IdSpill,Substance),
\+A=cancel_spill(IdSpill,IdAgent), \+A=delete_agent(IdAgent), poss(A,S).


%%%%%%%%agent_spills_total_size(IdAgent,Size)
holds(agent_spills_total_size(IdAgent,SizeTotal),do(A,S)) :-
A = make_spill(IdSpill,IdAgent), holds(spill_total_size(IdSpill,SizeAdd),S),
holds(agent_spills_total_size(IdAgent,SizePre),S), SizeTotal is SizeAdd +
SizePre, poss(A,S);

A = add_substance(IdSpill,Substance,Quantity),
holds(agent_spill(IdAgent,IdSpill),S),
holds(agent_spills_total_size(IdAgent,SizePre),S), SizeTotal is SizePre+Quantity,
poss(A,S);

A = del_substance(IdSpill,Substance,Q), holds(agent_spill(IdAgent,IdSpill),S),
holds(agent_spills_total_size(IdAgent,SizePre),S), SizeTotal is SizePre-Q,
poss(A,S);

A = del_total_substance(IdSpill,Substance),
holds(agent_spill(IdAgent,IdSpill),S),
holds(substance_spill_quantity(IdSpill,Substance,SpillTS),S),
holds(agent_spills_total_size(IdAgent,AgSTS),S), SizeTotal is AgSTS-SpillTS,
poss(A,S);

A = cancel_spill(IdSpill,IdAgent), holds(spill_total_size(IdSpill,SizeDel),S),
holds(agent_spills_total_size(IdAgent,SizePre),S), SizeTotal is SizeDel -
SizePre, poss(A,S);

holds(agent_spills_total_size(IdAgent,SizeTotal),S), A =
del_substance(IdSpill,Substance,Q), \+holds(agent_spill(IdAgent,IdSpill),S),
poss(A,S);

holds(agent_spills_total_size(IdAgent,SizeTotal),S), A =
del_substance(IdSpill,Substance,0), poss(A,S);
```

```
holds(agent_spills_total_size(IdAgent,SizeTotal),S), A =
del_total_substance(IdSpill,Substance), \+holds(agent_spill(IdAgent,IdSpill),S),
poss(A,S);

holds(agent_spills_total_size(IdAgent,SizeTotal),S), A =
add_substance(IdSpill,Substance,Quantity),
\+holds(agent_spill(IdAgent,IdSpill),S), poss(A,S);

holds(agent_spills_total_size(IdAgent,SizeTotal),S), A =
add_substance(IdSpill,Substance,0), poss(A,S);

holds(agent_spills_total_size(IdAgent,SizeTotal),S),
\+A=make_spill(IdSpill,IdAgent), \+A=cancel_spill(IdSpill,IdAgent),
\+A=add_substance(IdSpill,Substance,Quantity),
\+A=del_substance(IdSpill,Substance,Q),
\+A=del_total_substance(IdSpill,Substance), \+A=delete_agent(IdAgent), poss(A,S).

%%%%%%%% When an agent is created its spills are inizialized to 0
holds(agent_spills_total_size(IdAgent,0),do(A,S)) :-
A = new_agent(IdAgent), poss(A,S).


%%%%%%%%%best_place_to_spill(IdSpill,SpillPlace,IdAgent)
holds(best_place_to_spill(IdSpill,SpillPlace,IdAgent),do(A,S)) :-
A = set_best_place_to_spill(IdSpill,SpillPlace,IdAgent), poss(A,S);
holds(best_place_to_spill(IdSpill,SpillPlace,IdAgent),S),
\+A=set_best_place_to_spill(IdSpill,SpillPlace2,IdAgent),
\+A=delete_agent(IdAgent), \+A=cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%dilution_comunicated(IdAgent,IdAgentEntity,Size)
holds(dilution_comunicated(IdAgent,IdAgent2,IdSpill,Size),do(A,S)) :-
A = communicate_dilution(IdAgent,IdAgent2,IdSpill,Size), poss(A,S) ;
holds(dilution_comunicated(IdAgent,IdAgent2,Size),S), \+A=delete_agent(IdAgent),
\+A=cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%spill_diluted(IdSpill,Dilution)
holds(spill_diluted(IdSpill,Dilution),do(A,S)) :-
A = dilute_spill(IdSpill,Dilution), Dilution>0, poss(A,S) ;
holds(spill_diluted(IdSpill,Dilution),S), \+ A = delete_agent(IdAgent),
\+A=cancel_spill(IdSpill,IdAgent), \+A=spill_diluted(IdSpill,Dilution),
poss(A,S).

%%%%%%%%%pollutant_activity(Activity)
holds(pollutant_activity(Activity),do(A,S)) :-
A = set_pollutant_activity(Activity), poss(A,S);
holds(pollutant_activity(Activity),S) , \+A=unset_pollutant_activity(Activity),
poss(A,S).

%%%%%%%%%pollutant_agent(IdAgent)
holds(pollutant_agent(IdAgent),do(A,S)) :-
A = set_pollutant_activity(Activity), holds(agent_activity(IdAgent,Activity),S) ,
poss(A,S);
A = set_agent_activity(IdAgent,Activity), holds(pollutant_activity(Activity),S) ,
poss(A,S);
```

```
holds(pollutant_agent(IdAgent),S), A = unset_pollutant_activity(Activity),
holds(agent_activity(IdAgent,Activity2),S), Acitivity\=Activity2, poss(A,S);

holds(pollutant_agent(IdAgent),S), A = set_agent_activity(IdAgent,Activity),
holds(pollutant_activity(Activity),S), poss(A,S);

holds(pollutant_agent(IdAgent),S) ,\+ A = set_agent_activity(IdAgent,Activity),\+
A = unset_pollutant_activity(Activity), \+ A = delete_agent(IdAgent), poss(A,S).

%%%%%%%%%spill_authorized(IdSpill,SpillPlace,IdAgent,IdAgentEntity)
holds(spill_authorized(IdSpill,SpillPlace,IdAgent,IdAgentEntity),do(A,S)) :-
A = set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,positive,SpillPlace),
poss(A,S);
holds(spill_authorized(IdSpill,SpillPlace,IdAgent,IdAgentEntity),S),
\+A=set_spill_authorization(IdSpill,SpillPlace,IdAgent,IdAgentEntity,negative,Spi
llPlace), \+ A = delete_agent(IdAgent), \+ A = cancel_spill(IdSpill,IdAgent),
poss(A,S).

%%%%%%%%%spill_not_possible(IdAgent,IdSpill,SpillPlace)
holds(spill_not_possible(IdAgent,IdSpill,SpillPlace),do(A,S)) :-
A = set_spill_not_possible(IdAgent,IdSpill,SpillPlace) , poss(A,S);
holds(spill_not_possible(IdAgent,IdSpill,SpillPlace),S),\+ A =
set_spill_possible(IdAgent,IdSpill,SpillPlace), \+ A = delete_agent(IdAgent), \+
A = cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%spill_possible(IdAgent,IdSpill,SpillPlace)
holds(spill_possible(IdAgent,IdSpill,SpillPlace),do(A,S)) :-
A = set_spill_possible(IdAgent,IdSpill,SpillPlace), poss(A,S);
holds(spill_possible(IdAgent,IdSpill,SpillPlace),S),\+ A =
set_spill_not_possible(IdAgent,IdSpill,SpillPlace), \+ A = delete_agent(IdAgent),
\+ A = cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%spill_place(IdSpill,SpillPlace)
holds(spill_place(IdSpill,SpillPlace),do(A,S)) :-
A = set_spill_place(IdSpill,SpillPlace), poss(A,S);
holds(spill_place(IdSpill,SpillPlace),S),\+ A =
set_spill_place(IdSpill,SpillPlace2), \+ A = cancel_spill(IdSpill,IdAgent),
poss(A,S).

%%%%%%%%%spill_registered(IdSpill,RegisterPlace, IdAgentEntity)
holds(spill_registered(IdSpill,census, IdAgentEntity),do(A,S)) :-
A = register_spill(IdSpill,census, IdAgentEntity), poss(A,S);
holds(spill_registered(IdSpill,census, IdAgentEntity),S), \+ A =
delete_agent(IdAgentEntity), \+ A = cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%spill_total_size(IdSpill,SizeS)
holds(spill_total_size(IdSpill,SizeT),do(A,S)) :-
A =
add_substance(IdSpill,Substance,SizeA),holds(spill_total_size(IdSpill,SizeP),S) ,
SizeT is SizeA + SizeP, poss(A,S);
A =
del_substance(IdSpill,Substance,SizeD),holds(spill_total_size(IdSpill,SizeP),S) ,
SizeT is SizeP - SizeD, poss(A,S);
A =
del_total_substance(IdSpill,Substance),holds(substance_spill_quantity(IdSpill,Sub
```

```
stance,SizeD),S) , holds(spill_total_size(IdSpill,SizeP),S) , SizeT is SizeP -
SizeD, poss(A,S);

holds(spill_total_size(IdSpill,SizeT),S),\+ A =
add_substance(IdSpill,Substance,Quantity),\+ A =
del_substance(IdSpill,Substance,SizeD),\+ A =
del_total_substance(IdSpill,Substance),\+ A = cancel_spill(IdSpill,IdAgent),
poss(A,S).

%%%%%%%%
holds(spill_total_size(IdSpill,0),do(A,S)) :-
A = create_spill(IdSpill), poss(A,S).

%%%%%%%%%spill_type(IdSpill,SpillType)
holds(spill_type(IdSpill,SpillType),do(A,S)) :-
A = set_spill_type(IdSpill,SpillType), poss(A,S);
holds(spill_type(IdSpill,SpillType),S),\+ A = set_spill_type(IdSpill,SpillType2),
\+ A = cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%spill_contains_limited_substance(IdSpill,Substance)
holds(spill_contains_limited_substance(IdSpill,Substance),do(A,S)) :-
A = add_substance(IdSpill,Substance,Quantity),
holds(substance_limitation(Substance,Limit),S), poss(A,S) ;

holds(spill_contains_limited_substance(IdSpill,Substance),S), A =
del_substance(IdSpill,Substance,QDel)
,holds(substance_spill_quantity(IdSpill,Substance,QPre),S), QPre>QDel, poss(A,S);

holds(spill_contains_limited_substance(IdSpill,Substance),S), \+ A =
del_substance(IdSpill,Substance,QuantityD), \+ A =
del_total_substance(IdSpill,Substance), \+ A = cancel_spill(IdSpill,IdAgent),
poss(A,S).

%%%%%%%%%substance_limitation(Substance,Limit)
holds(substance_limitation(Substance,Limit),do(A,S)) :-
A = set_substance_limitation(Substance,Limit), poss(A,S);
holds(substance_limitation(Substance,Limit),S),\+ A =
set_substance_limitation(Substance,Limit2), poss(A,S).

%%%%%%%%%substance_spill_quantity(IdSpill,Substance,Quantity)
holds(substance_spill_quantity(IdSpill,Substance,QuantityT),do(A,S)) :-
A = add_substance(IdSpill,Substance,QuantityA) ,
holds(substance_spill_quantity(IdSpill,Substance,QuantityP),S) , QuantityT is
QuantityP+QuantityA, poss(A,S);
A = add_substance(IdSpill,Substance,QuantityT) ,
\+holds(substance_spill_quantity(IdSpill,Substance,QuantityP),S), poss(A,S);

A = del_substance(IdSpill,Substance,QuantityD),
holds(substance_spill_quantity(IdSpill,Substance,QuantityP),S) , QuantityT is
QuantityP-QuantityD, poss(A,S);

holds(substance_spill_quantity(IdSpill,Substance,QuantityT),S), A =
add_substance(IdSpill,Substance2,QuantityA) , Substance\=Substance2, poss(A,S);

holds(substance_spill_quantity(IdSpill,Substance,QuantityT),S),
\+ A = add_substance(IdSpill,Substance2,QuantityA) ,
```

```
\+ A = del_substance(IdSpill,Substance,QuantityD) ,
\+ A = cancel_spill(IdSpill,IdAgent) ,
\+ A = del_total_substance(IdSpill,Substance), poss(A,S).

holds(substance_spill_quantity(IdSpill,Substance,0),do(A,S)) :-
A = del_total_substance(IdSpill,Substance), poss(A,S);
A = create_spill(IdSpill), substance(Substance), poss(A,S).

%%%%%%%%%spill_respects_limitations(IdSpill)
holds(spill_respects_limitations(IdSpill),S) :-
\+ holds(spill_violates_limitation(IdSpill,Substance),S),holds(spill(IdSpill),S).

%%%%%%%%%spill_violates_limitation(IdSpill,Substance)
holds(spill_violates_limitation(IdSpill,Substance),do(A,S)) :-
A = add_substance(IdSpill,Substance,QuantityA),
holds(substance_spill_quantity(IdSpill,Substance,QuantityP),S) ,
holds(substance_limitation(Substance,Limit),S) , Limit < QuantityP+QuantityA,
poss(A,S);

A = add_substance(IdSpill,Substance,QuantityA),
\+holds(substance_spill_quantity(IdSpill,Substance,QuantityP),S) ,
holds(substance_limitation(Substance,Limit),S) , Limit < QuantityA, poss(A,S);

A = set_substance_limitation(Substance,Limit) ,
holds(substance_spill_quantity(IdSpill,Substance,QuantityP),S), QuantityP>Limit,
poss(A,S);

holds(spill_violates_limitation(IdSpill,Substance),S), A =
del_substance(IdSpill,Substance,QuantityD) ,
holds(substance_spill_quantity(IdSpill,Substance,QuantityP),S) ,
holds(substance_limitation(Substance,Limit),S) , Limit < QuantityP-QuantityD,
poss(A,S);

holds(spill_violates_limitation(IdSpill,Substance),S), \+ A =
del_total_substance(IdSpill,Substance), \+ A =
del_substance(IdSpill,Substance,QuantityD), \+ A =
set_substance_limitation(Substance,Limit), \+ A = cancel_spill(IdSpill,IdAgent) ,
poss(A,S).

%%%NORMS

%%%%%%%%%norm(71,IdAgent)
holds(norm(71,IdAgent),do(A,S)):-
A = make_spill(IdSpill,IdAgent) , holds(spill_total_size(IdSpill,SizeS),S) ,
holds(agent_spills_total_size(IdAgent,SizeA),S) , Total is SizeA+SizeS ,
Total>6000,\+holds(norm(71,IdAgent),S),poss(A,S) ;
A = add_substance(IdSpill,Sub,Qua), holds(agent_spill(IdAgent,IdSpill),S),
holds(agent_spills_total_size(IdAgent,Size),S),
6000<Qua+Size,\+holds(norm(71,IdAgent),S), poss(A,S);

A = set_agent_type(IdAgent,non_domestic) ,
holds(pollutant_agent(IdAgent),S),\+holds(norm(71,IdAgent),S) , poss(A,S);
A = set_agent_activity(IdAgent,Activity) , holds(pollutant_activity(Activity),S)
, holds(agent_type(IdAgent,non_domestic),S),\+holds(norm(71,IdAgent),S) ,
poss(A,S);
```

```
A = set_pollutant_activity(Activity) , holds(agent_activity(IdAgent,Activity),S),
holds(agent_type(IdAgent,non_domestic),S),\+holds(norm(71,IdAgent),S) ,
poss(A,S);

holds(norm(71,IdAgent),S), A = set_agent_type(IdAgent,non_domestic), poss(A,S);

holds(norm(71,IdAgent),S), A = set_agent_type(IdAgent,AgentType),
AgentType\=non_domestic,holds(agent_spills_total_size(IdAgent,SizeA),S),
SizeA>6000, poss(A,S);

holds(norm(71,IdAgent),S), A = set_agent_activity(IdAgent,Activity),
holds(pollutant_activity(Activity),S), poss(A,S);

holds(norm(71,IdAgent),S), A = set_agent_activity(IdAgent,Activity),
\+holds(pollutant_activity(Activity),S),holds(agent_spills_total_size(IdAgent,Siz
eA),S), SizeA>6000, poss(A,S);

holds(norm(71,IdAgent),S), A = unset_pollutant_activity(Activity) ,
holds(agent_activity(IdAgent,Activity),S),holds(agent_spills_total_size(IdAgent,S
izeA),S), SizeA>6000, poss(A,S);

holds(norm(71,IdAgent),S), A = unset_pollutant_activity(Activity) ,
\+holds(agent_activity(IdAgent,Activity),S), poss(A,S);

holds(norm(71,IdAgent),S), A = cancel_spill(IdSpill,IdAgent),
holds(agent_type(IdAgent,non_domestic),S),holds(pollutant_agent(IdAgent),S),
poss(A,S);

holds(norm(71,IdAgent),S), A = cancel_spill(IdSpill,IdAgent),
holds(spill_total_size(IdSpill,SizeS),S),
holds(agent_spills_total_size(IdAgent,SizeA),S), VAR is SizeA -SizeS, VAR> 6000,
poss(A,S);

holds(norm(71,IdAgent),S), A = del_substance(IdSpill,Sub,Qu),
\+holds(agent_spill(IdAgent,IdSpill),S), poss(A,S);

holds(norm(71,IdAgent),S), A =
del_substance(IdSpill,Sub,Qu),holds(agent_spill(IdAgent,IdSpill),S),
holds(agent_type(IdAgent,non_domestic),S),holds(pollutant_agent(IdAgent),S),
poss(A,S);

holds(norm(71,IdAgent),S), A =
del_substance(IdSpill,Sub,Qu),holds(agent_spill(IdAgent,IdSpill),S),
holds(agent_spills_total_size(IdAgent,SizeA),S), VAR is SizeA -Qu, VAR > 6000,
poss(A,S);

holds(norm(71,IdAgent),S), A = del_total_substance(IdSpill,Sub),
\+holds(agent_spill(IdAgent,IdSpill),S), poss(A,S);

holds(norm(71,IdAgent),S), A =
del_total_substance(IdSpill,Sub),holds(agent_spill(IdAgent,IdSpill),S),holds(agen
t_type(IdAgent,non_domestic),S), holds(pollutant_agent(IdAgent),S), poss(A,S);

holds(norm(71,IdAgent),S), A =
del_total_substance(IdSpill,Sub),holds(agent_spill(IdAgent,IdSpill),S),
```

```
holds(agent_spills_total_size(IdAgent,SizeA),S),
holds(spill_total_size(IdSpill,Qu),S), VAR is SizeA -Qu, VAR > 6000, poss(A,S);

holds(norm(71,IdAgent),S),\+ A = set_agent_type(IdAgent,Type) , \+ A =
set_agent_activity(IdAgent,Activity), \+A =
unset_pollutant_activity(Activity2),\+ A = del_total_substance(IdSpill,Sub), \+ A
= del_substance(IdSpill,Sub,Qu), \+ A = delete_agent(IdAgent),\+ A =
cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%norm(72,IdAgent)
holds(norm(72,IdAgent),do(A,S)) :-
A =
set_agent_type(IdAgent,domestic),holds(agent(IdAgent),S),\+holds(norm(72,IdAgent)
,S), poss(A,S);

holds(norm(72,IdAgent),S), A = set_agent_type(IdAgent,domestic), poss(A,S);

holds(norm(72,IdAgent),S),\+ A = set_agent_type(IdAgent,Type), \+ A =
delete_agent(IdAgent), poss(A,S).

%%%%%%%%%norm(731,IdAgent)
holds(norm(731,IdAgent,IdSpill),S):-
holds(agent(IdAgent),S),holds(spill(IdSpill),S),
holds(agent_spill(IdAgent,IdSpill),S), \+holds(norm(732,IdAgent,IdSpill),S).

%%%%%%%%%norm(732,IdAgent,IdSpill)
holds(norm(732,IdAgent,IdSpill),do(A,S)):-
A = set_best_place_to_spill(IdSpill,environment,IdAgentEntity) ,
holds(agent_associated_entity(IdAgent,IdAgentEntity),S),\+holds(norm(732,IdAgent,
IdSpill),S),poss(A,S) ;

A = set_associated_entity(IdAgent,IdAgentEntity) ,
holds(best_place_to_spill(IdSpill,environment,IdAgentEntity),S),\+holds(norm(732,
IdAgent,IdSpill),S), poss(A,S);

holds(norm(732,IdAgent,IdSpill),S), A =
set_best_place_to_spill(IdSpill,environment,IdAgent2), poss(A,S);

holds(norm(732,IdAgent,IdSpill),S), A =
set_best_place_to_spill(IdSpill,Place,IdAgentEntity), Place\=environment,
\+holds(agent_associated_entity(IdAgent,IdAgentEntity),S),poss(A,S) ;

holds(norm(732,IdAgent,IdSpill),S), A =
set_associated_entity(IdAgent,IdAgentEntity2),
holds(best_place_to_spill(IdSpill,environment,IdAgentEntity2),S),poss(A,S) ;

holds(norm(732,IdAgent,IdSpill),S), \+ A =
set_best_place_to_spill(IdSpill,SpillPlace,IdAgentEntity), \+ A =
set_associated_entity(IdAgent,IdAgentEntity2), \+ A = delete_agent(IdAgent), \+ A
= cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%norm(74,IdAgent,IdSpill)
holds(norm(74,IdAgent,IdSpill),do(A,S)):-
A = make_spill(IdSpill,IdAgent) ,
holds(agent_associated_entity(IdAgent,IdAgentEntity),S) ,
```

```
\+holds(spill_registered(IdSpill,census,
IdAgentEntity),S),\+holds(norm(74,IdAgent,IdSpill),S), poss(A,S);

holds(norm(74,IdAgent,IdSpill),S), A = register_spill(IdSpill,census,
IdAgentEntity) , \+holds(agent_associated_entity(IdAgent,IdAgentEntity),S),
poss(A,S);

holds(norm(74,IdAgent,IdSpill),S), A =
set_associated_entity(IdAgent,IdAgentEntity), \+
holds(spill_registered(IdSpill,census, IdAgentEntity),S), poss(A,S);

holds(norm(74,IdAgent,IdSpill),S), \+ A = register_spill(IdSpill,Place,
IdAgentEntity),\+ A = set_associated_entity(IdAgent,IdAgentEntity2), \+ A =
delete_agent(IdAgent), \+ A = cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%norm(81,IdAgent)
holds(norm(81,IdAgent),S):-holds(agent(IdAgent),S).

%%%%%%%%%norm(8121,IdAgent,IdSpill)
holds(norm(8121,IdAgent,IdSpill),S):-
holds(agent(IdAgent),S),holds(spill(IdSpill),S),
holds(agent_spill(IdAgent,IdSpill),S), \+holds(norm(8122,IdAgent,IdSpill),S).

%%%%%%%%%norm(8122,IdAgent)
holds(norm(8122,IdAgent,IdSpill),do(A,S)):-
A = set_agent_situation(IdAgent,risk) ,
holds(dilution_comunicated(IdAgent,IdAgentEntity,IdSpill,Size),S) ,
holds(agent_associated_entity(IdAgent,IdAgentEntity),S),
\+holds(norm(8122,IdAgent,IdSpill),S), poss(A,S);
A = set_agent_situation(IdAgent,emergency) ,
holds(dilution_comunicated(IdAgent,IdAgentEntity,IdSpill,Size),S),
holds(agent_associated_entity(IdAgent,IdAgentEntity),S),
\+holds(norm(8122,IdAgent,IdSpill),S), poss(A,S);

A = communicate_dilution(IdAgent,IdAgentEntity,IdSpill,Size) ,
holds(agent_associated_entity(IdAgent,IdAgentEntity),S) ,
holds(agent_situation(IdAgent,risk),S),\+holds(norm(8122,IdAgent,IdSpill),S),
poss(A,S);
A = communicate_dilution(IdAgent,IdAgentEntity,IdSpill,Size) ,
holds(agent_associated_entity(IdAgent,IdAgentEntity),S) ,
holds(agent_situation(IdAgent,emergency),S),
\+holds(norm(8122,IdAgent,IdSpill),S), poss(A,S);

A = set_associated_entity(IdAgent,IdAgentEntity) ,
holds(dilution_comunicated(IdAgent,IdAgentEntity,IdSpill,Size),S) ,
holds(agent_situation(IdAgent,risk),S),\+holds(norm(8122,IdAgent,IdSpill),S),
poss(A,S);
A = set_associated_entity(IdAgent,IdAgentEntity) ,
holds(dilution_comunicated(IdAgent,IdAgentEntity,IdSpill,Size),S) ,
holds(agent_situation(IdAgent,emergency),S),
\+holds(norm(8122,IdAgent,IdSpill),S), poss(A,S);

holds(norm(8122,IdAgent,IdSpill),S), A = set_agent_situation(IdAgent,risk),
poss(A,S);
```

```
holds(norm(8122,IdAgent,IdSpill),S), A = set_agent_situation(IdAgent,emmergency),
poss(A,S);

holds(norm(8122,IdAgent,IdSpill),S),\+ A = set_agent_situation(IdAgent,Sit),
\+A=delete_agent(IdAgent), \+ A = cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%norm(8131,IdAgent,IdSpill)
holds(norm(8131,IdAgent,IdSpill),S) :-
holds(agent(IdAgent),S),holds(spill(IdSpill),S),
holds(spill_type(IdSpill,white_waters),S), \+
holds(norm(8132,IdAgent,IdSpill),S).

%%%%%%%%%norm(8132,IdAgent,IdSpill)
holds(norm(8132,IdAgent,IdSpill),do(A,S)):-
A = set_spill_not_possible(IdAgent,IdSpill,separative_net) ,
holds(spill_type(IdSpill,white_waters),S),
holds(spill_not_possible(IdAgent,IdSpill,environment),S) ,
holds(spill_authorized(IdSpill,sewer_system,IdAgent,IdAgentEntity),S) ,
holds(agent_associated_entity(IdAgent,IdAgentEntity),S),
\+holds(norm(8132,IdAgent,IdSpill),S) , poss(A,S);

A = set_spill_not_possible(IdAgent,IdSpill,environment) ,
holds(spill_type(IdSpill,white_waters),S),
holds(spill_not_possible(IdAgent,IdSpill,separative_net),S) ,
holds(spill_authorized(IdSpill,sewer_system,IdAgent,IdAgentEntity),S) ,
holds(agent_associated_entity(IdAgent,IdAgentEntity),S),
\+holds(norm(8132,IdAgent,IdSpill),S), poss(A,S) ;

A = set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,positive,sewer_system),
holds(spill_type(IdSpill,white_waters),S),
holds(agent_associated_entity(IdAgent,IdAgentEntity),S) ,
holds(spill_not_possible(IdAgent,IdSpill,environment),S) ,
holds(spill_not_possible(IdAgent,IdSpill,separative_net),S),
\+holds(norm(8132,IdAgent,IdSpill),S), poss(A,S);

A = set_associated_entity(IdAgent,IdAgentEntity),
holds(spill_type(IdSpill,white_waters),S),
holds(spill_authorized(IdSpill,sewer_system,IdAgent,IdAgentEntity),S) ,
holds(spill_not_possible(IdAgent,IdSpill,environment),S) ,
holds(spill_not_possible(IdAgent,IdSpill,separative_net),S),
\+holds(norm(8132,IdAgent,IdSpill),S) ,poss(A,S) ;

holds(norm(8132,IdAgent,IdSpill),S), A =
set_spill_possible(IdAgent,IdSpill,sewer_system), poss(A,S);

holds(norm(8132,IdAgent,IdSpill),S), A =
set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,positive,Place), poss(A,S);

holds(norm(8132,IdAgent,IdSpill),S), A =
set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,Auth,Place),
Place\=sewer_system, poss(A,S);

holds(norm(8132,IdAgent,IdSpill),S), A =
set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,negative,sewer_system),
holds(agent_associated_entity(IdAgent,IdAgentEntity2),S),
IdAgentEntity\=IdAgentEntity2, poss(A,S);
```

```
holds(norm(8132,IdAgent,IdSpill),S), A = set_spill_type(IdSpill,white_waters),
poss(A,S);

holds(norm(8132,IdAgent,IdSpill),S),\+ A =
set_spill_possible(IdAgent,IdSpill,Place),\+ A =
set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,Auth,Place),\+A =
set_spill_type(IdSpill,Type), \+ A = delete_agent(IdAgent), \+ A =
cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%norm(82,IdAgent,IdSpill)
holds(norm(82,IdAgent,IdSpill),do(A,S)) :-
A = set_agent_type(IdAgent,domestic),
holds(spill_contains_limited_substance(IdSpill,Substance),S),
\+holds(norm(82,IdAgent,IdSpill),S), poss(A,S);

A = add_substance(IdSpill,Substance,Quantity) ,
holds(agent_type(IdAgent,domestic),S) , holds(agent_spill(IdAgent,IdSpill),S) ,
holds(substance_limitation(Substance,Limit),S) , Limit > 0 , Quantity > 0,
\+holds(norm(82,IdAgent,IdSpill),S), poss(A,S);

holds(norm(82,IdAgent,IdSpill),S), A = set_agent_type(IdAgent,domestic),
poss(A,S) ;

holds(norm(82,IdAgent,IdSpill),S), A = del_substance(IdSpill,Substance,Quantity),
holds(spill_contains_limited_substance(IdSpill,Substance2),S),
Substance\=Substance2, poss(A,S);

holds(norm(82,IdAgent,IdSpill),S), A = del_substance(IdSpill,Substance,QD),
holds(substance_spill_quantity(IdSpill,Substance,QPre),S), QPre>QD,poss(A,S) ;

holds(norm(82,IdAgent,IdSpill),S), A = del_total_substance(IdSpill,Substance),
holds(spill_contains_limited_substance(IdSpill,Substance2),S),
Substance\=Substance2, poss(A,S);

holds(norm(82,IdAgent,IdSpill),S) , \+ A = set_agent_type(IdAgent,Type),
\+A=del_substance(IdSpill,Sub,Q), \+ A = del_total_substance(IdSpill,Substance),
\+A=delete_agent(IdAgent), \+ A = cancel_spill(IdSpill,IdAgent), poss(A,S).

%%%%%%%%%norm(1021,IdAgent,IdSpill)
holds(norm(1021,IdAgent,IdSpill),S):-
holds(agent(IdAgent),S),holds(spill(IdSpill),S),
holds(agent_spill(IdAgent,IdSpill),S), \+holds(norm(1022,IdAgent,IdSpill),S).

%%%%%%%%%norm(1022,IdAgent,IdSpill)
holds(norm(1022,IdAgent,IdSpill),do(A,S)):-
A = set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,positive,sewer_system)
, holds(agent_associated_entity(IdAgent,IdAgentEntity),S) ,
holds(spill_type(IdSpill,black_waters),S) ,
holds(spill_respects_limitations(IdSpill),S),
\+holds(norm(1022,IdAgent,IdSpill),S), poss(A,S);

A = set_associated_entity(IdAgent,IdAgentEntity) ,
holds(spill_authorized(IdSpill,sewer_system,IdAgent,IdAgentEntity),S) ,
holds(spill_type(IdSpill,black_waters),S) ,
```

```
holds(spill_respects_limitations(IdSpill),S),
\+holds(norm(1022,IdAgent,IdSpill),S), poss(A,S);

(A = del_substance(IdSpill,Substance,QuantityD) ,
holds(substance_limitation(Substance,Limit),S),
holds(substance_spill_quantity(IdSpill,Substance,QuantityB),S),
\+holds(agent_limited_substance(IdAgent,Substance2,Quantity),S) ,
Substance\=Substance2,
Limit >QuantityB-QuantityD),
holds(spill_authorized(IdSpill,sewer_system,IdAgent,IdAgentEntity),S) ,
holds(spill_type(IdSpill,black_waters),S),
\+holds(norm(1022,IdAgent,IdSpill),S),poss(A,S) ;

A = del_total_substance(IdSpill,Substance,QuantityD) ,
holds(agent_limited_substance(IdAgent,Substance,Quantity),S) ,
\+holds(agent_limited_substance(IdAgent,Substance2,Quantity),S) ,
Substance\=Substance2,
holds(spill_authorized(IdSpill,sewer_system,IdAgent,IdAgentEntity),S) ,
holds(spill_type(IdSpill,black_waters),S),
\+holds(norm(1022,IdAgent,IdSpill),S), poss(A,S);

holds(norm(1022,IdAgent,IdSpill),S), A =
set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,positive,SpillPlace),
poss(A,S);

holds(norm(1022,IdAgent,IdSpill),S), A =
set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,Authorization,SpillPlace),
SpillPlace\=sewer_system, poss(A,S);

holds(norm(1022,IdAgent,IdSpill),S), A =
set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,Authorization,SpillPlace),
\+holds(holds(agent_associated_entity(IdAgent,IdAgentEntity),S)), poss(A,S);

holds(norm(1022,IdAgent,IdSpill),S), A =
add_substance(IdSpill,Substance,AddedQuantity),
\+holds(substance_limitation(Substance,Limit),S), poss(A,S);

holds(norm(1022,IdAgent,IdSpill),S), A =
add_substance(IdSpill,Substance,AddedQuantity),
\+holds(substance_spill_quantity(IdSpill,Substance,Base),S),
holds(substance_limitation(Substance,Limit),S), Limit>AddedQuantity, poss(A,S);

holds(norm(1022,IdAgent,IdSpill),S), A =
add_substance(IdSpill,Substance,AddedQuantity),
holds(substance_spill_quantity(IdSpill,Substance,BaseQuantity),S),
holds(substance_limitation(Substance,Limit),S), Limit>AddedQuantity +
BaseQuantity, poss(A,S);

holds(norm(1022,IdAgent,IdSpill),S),\+ A =
set_spill_authorization(IdSpill,IdAgent,IdAgentEntity,Auth,Place) ,\+ A =
add_substance(IdSpill,Substance,AddedQuantity), \+ A = delete_agent(IdAgent),
poss(A,S).

%%%CONSTANTS

authorization(positive).
```

```
authorization(negative).
situation(risk).
situation(emergency).
situation(normal).

agentType(non_domestic).
agentType(domestic).
agentType(wca).
agentType(wla).

spillPlace(sewer_system).
spillPlace(environment).
spillPlace(separative_net).

spillType(black_waters).
spillType(white_waters).

registerPlace(census).

substance(mes).
substance(dbo5).
substance(dqo).
substance(olis).
substance(greixos).
substance(clorur).
substance(conductivity).
substance(solid_matter).
substance(viscous_matter).
substance(solvents).
substance(other).

activity(chemical).
activity(pharmaceutical).

%%%INICIAL SITUATION

holds(agent(ag1),s0).
holds(agent_situation(ag1,normal),s0).
holds(agent_type(ag1,non_domestic),s0).
holds(agent_activity(ag1,pharmaceutical),s0).
holds(agent_spills_total_size(ag1,1900),s0).

holds(entity_agent(ent1),s0).
holds(agent_type(ent1,wla),s0).

holds(entity_agent(ent2),s0).
holds(agent_type(ent2,wla),s0).

holds(agent_associated_entity(ag1,ent1),s0).

holds(agent(ag2),s0).
holds(agent_situation(ag2,normal),s0).
holds(agent_type(ag2,domestic),s0).
holds(agent_activity(ag2,chemical),s0).
holds(agent_spills_total_size(ag2,0),s0).
holds(agent_spill(ag1,sp1),s0).
```

```
holds(pollutant_activity(pharmaceutical),s0).
holds(pollutant_agent(ag1),s0).

holds(spill(sp1),s0).
holds(spill_total_size(sp1,1900),s0).
holds(spill_place(sp1,sewer_system),s0).
holds(spill_type(sp1,black_waters),s0).
holds(substance_spill_quantity(sp1,mes,1000),s0).
holds(substance_spill_quantity(sp1,dbo5,100),s0).
holds(substance_spill_quantity(sp1,dqo,800),s0).

holds(spill_violates_limitation(sp1,mes),s0).

holds(agent_limited_substance(ag1,mes,1000),s0).
holds(agent_limited_substance(ag1,dbo5,100),s0).
holds(agent_limited_substance(ag1,dqo,800),s0).


holds(substance_limitation(mes,750),s0).
holds(substance_limitation(dbo5,750),s0).
holds(substance_limitation(dqo,1500),s0).
holds(substance_limitation(olis,250),s0).
holds(substance_limitation(greixos,250),s0).
holds(substance_limitation(clorur,2500),s0).
holds(substance_limitation(conductivity,6000),s0).
holds(substance_limitation(solid_matter,0),s0).
holds(substance_limitation(viscous_matter,0),s0).
holds(substance_limitation(solvents,0),s0).

holds(norm(71,ag1),s0).
holds(norm(72,ag2),s0).
holds(norm(74,ag1,sp1),s0).
```

15 Annex VI: Scholarly paper

Paper sent and accepted in the 8th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)

Using Situation Calculus for Normative Agents in Urban Wastewater Systems

Juan Carlos Nieves, Dario Garcia, Montse Aulinas, Ulises Cortés

Abstract Water quality management policies on a river basin scale are of special importance in order to prevent and/or reduce pollution of several human sources into the environment. Industrial effluents represent a priority issue particularly in urban wastewater systems that receive mixed household and industrial wastewaters, apart from rainfall water. In this paper, we present an analysis and an implementation of normative agents which capture concrete regulations of the Catalan pollution-prevention policies. The implementation of the normative agents is based on situation calculus.

Key words: Rational Agents, Environmental Decision Support Systems, practical normative reasoning, situation calculus.

1 Introduction

Environmental decision-making is a complex, multidisciplinary and crucial task. As an example, in the water management field, water managers have to deal with complex problems due to the characteristics of processes that occur within environmental systems. In addition to this, water managers have to deal with normative regulations that have to be considered in any decision.

In particular, at European level, Directive 96/61/EC on Integrated Pollution Prevention and Control (IPPC) (CEC, 1996) [4] was developed to apply an integrated environmental approach to the regulation of certain industrial activities. This means that, at least, emissions to air, water (including discharges to sewer) and land must be considered together. It also means that regulators must set permit conditions so as

Juan Carlos Nieves, Dario Garcia, Ulises Cortés
Universitat Politècnica de Catalunya, C/Jordi Girona 1-3, E08034, Barcelona, Spain, e-mail: {jcnieves,ia}@lsi.upc.edu and dariogarcia@gmail.com

Montse Aulinas
Laboratory of Chemical and Environmental Engineering, University of Girona,, Spain, e-mail: aulinas@lequia.udg.cat

to achieve a high level of protection for the environment as a whole. Several national and regional efforts are being done in order to improve water quality management as well as to accomplish European regulations. Concretely, we analyse the Catalan experience as a realistic example of adapting European guidelines to manage water taking into account the local/regional reality.

In order to analyse the context of pollution-prevention policies in Catalonia we take a concrete regulation: the Decree 130/2003 [11]. It is a regional regulation appeared as a consequence of the Catalan sanitation programme. One of the aims of the updated programme is to directly link the urban wastewater treatment program with the industrial wastewater treatment program. It pays special attention to the industrial component of urban WWTPs in order to facilitate the connection to the public system of those industries and/or industrial parks that accomplish the requirements, and that is the reason why Decree 130/2003 was developed.

Following the perspective that a software agent is any active entity whose behavior is described by mental notions such as knowledge, goals, abilities, commitments, *etc.*, we have been exploring the definition of intelligent agents provided by a ***normative knowledge*** in order to manage concrete normative regulations which are in the context of UWS.

A central issue of a successful normative agent implementation is the selection of a formalism for performing *practical normative reasoning*. Although several powerful formalisms exist, finding the right one is a non-trivial challenge, as it must provide a level of expressiveness that serves the practical problems at hand in a tractable way. In the literature, one can find several approaches for performing normative reasoning such as deontic logic, temporal logic, dynamic logic, *etc.* [10]; however, these approaches have a high computational cost. One can agree that formal methods do help in the long run in helping to develop a clearer understanding of problems and solutions; hence, the definition of computable tractable approaches base on formal methods takes relevance for performing practical normative reasoning.

Situation calculus has shown to be a practical approach for facing real problems [1, 6, 8]. Moreover, situation calculus allows the specification of any set of complex action expressions. There are some results that have shown that situation calculus is flexible enough for performing normative reasoning [5].

In this paper, we describe the implementation of normative agents based on situation calculus for performing practical normative reasoning. The normative knowledge structure follows an approach introduced in [2] and explored in [12, 13]. Unlike the approach presented in [13] which extends the action language $\mathcal{A}$ for capturing norms, in this paper we explorer a norm's lifecycle by considering states of the world (situations/sets of fluents). This means that our main concern will be monitoring the states of norms such that the states of a norm can be: active, inactive and violated.

We will describe an analysis of a specific Catalan regulation for providing normative knowledge to normative agents. Also, we will describe the implementation of these normative agents by considering situation calculus.

The rest of the paper is divided as follows: In §2, a realistic hypothetical scenario is described in order to illustrate the role of some regulations for managing industrial

discharges. In §3, it is described, at a high level, *how* to introduce normative knowledge in a situation calculus specification. In §4, we describe *how* to implement the approach presented in §3. In the last section, we present our conclusions.

2 Realistic Scenario

In this section, a realistic hypothetical scenario is described in order to illustrate the role of some regulations for managing industrial discharges.

At the municipality of Ecopolis a new industry called MILK XXI pretends to set up. As a result of its production processes the main characteristics of its wastewater will be as follows:

- Flow: 60 l/s (5184 m^3/day)
- Suspended solids: 130 mg/l
- BOD5: 450 mg/l
- COD 800 mg/l
- Oils and greases : 275 mg/l

The Milk factory plans to work 16 h/day (two shifts), 225 days per year. It plans to get connection to the municipal sewer system that collects wastewater from a population of 12000 inhabitants and transports it to the municipal WWTP. WWTP complies strictly with regulations. The owner of the industry submits the request to obtain authorization to discharge into the municipal sewer system, which it is compulsory by law (Decree 130/2003 that settles the public sewer systems regulations). Moreover, Milk industry plans to apply BAT[1] in order to reduce the consumption of water, so it applies as well for the consideration of this fact into the final authorization decision.

Accordingly the industry intends to reduce 30 % on water consumption, and consequently the increment of pollutant concentrations, is projected to be as follows:

- Flow (reduction): 42 l/s (3628,8 m^3/day)
- Suspended solids: 200 mg/l
- BOD5: 600 mg/l
- COD 1000 mg/l
- Oils and greases : 357,5 mg/l

Several rules are launched to manage this case, from the regulation analyzed in this work. As follows we describe the immediate agents implied in the case as well as the type of norms involved:

[1] BAT: Best Available Techniques, (CEC 2006) [4].

Action	Agent	Type of Norm
Request authorization	Industry agent	Obligation (7.1)
Given authorization:	Water Catalan Agency Agent	
thresholds		Obligation (13.1)
average flow		Obligation (13.2)
conditions		Obligation (13.2)
exceptions		
etc.		
Apply BAT (declare this when requesting authorization)	Industry Agent	Obligation(8.3)
Discharge	Industrial Agent	Obligation (8.2)

Due to lack of space, we omit the detailed description of each agent; however, a version of those can be found in [3]. Observe that the behavior of each agent is fixed by the set of norms that it has to observe. In the following section, we are going to describe an approach for expressing these norms in the mental notions of an agent.

3 Normative Specification based on Situation Calculus

In here we are going to describe, at a high level, the modeling process of norms. Since environmental domains are dynamic domains, a dynamic domain is a domain where truth values change with time, the described approach deals with the specification of norms in dynamic domains. In particular, we have used Situation Calculus [8, 9], which works with the concepts of *fluents* and *actions*.

To formally specify any set of norms, first it is necessary to analyse the domain these norms will work with. In Situation Calculus, the world in a certain moment (represented by a first order term known as *situation*) is defined by the value of a set of predicates, known as *fluents*, in that moment. The change between situations is triggered by *actions*, which can be parameterized. Actions may change the value of one or more fluents, therefore modifying the current situation.

A constant s_0 is defined as the initial situation, for which fluent values are given as to define that state. The binary function $do(X,Y)$ denotes the situation resultant of executing action X in situation Y. The binary function $holds(Z,W)$ denotes the truth value of a fluent Z in a situation W. With the combination of these two functions we can represent all possible worlds and relationships between them, inherent of dynamic domains.

As any approach for temporal reasoning, Situation Calculus must deal with the *Frame Problem* to make its implementation consistent [9]. Aware of that, we present a specification that fully asserts the effects of all actions on every norm.

Before working on how to specify norms we analyse them, following the works of [2, 12, 13] and keeping situation calculus in mind. To fully specify a norm several aspects must be taken into account:

Type of norm: norms that oblige to do something, norms that allow/permit something or norms that forbid something. It is important to take special care with allowing norms, since to fully specify their content it seems to be necessary to split them into two norms, one that allows something and one that forbids something,

e.g., *"It is allowed to spill black waters into the river if one has the required authorization"*. This allowing norm implicitly includes the following forbidding norm: *"It is forbidden to spill black waters into the river without authorization"*.

Conditions and content: separate the norm conditions and the norm content in order to study the characteristics of situations, in which the norm is active and in which the norm is violated.

States: the set of variables the norm refers to. For each possible value of those variables the norm has a state, *e.g.*, if a norm is applied to an agent then it would have one variable such as *IdAgent*, and if it is applied to an agent's spill then it would have two variables, *IdAgent* and *IdSpill*.

Actions: a complete list of the domain's actions that may influence the activation state and the violation state, separately, of each norm.

Preconditions: define the preconditions for each action, that is, in which situations are they possible and the conditions about their parameters.

Now we can start to specify our norms. We follow Reiter's solution presented in [14] to our normative domain. We propose to split the specification in two parts corresponding to the main properties that all norms have:

- The scenarios in which the norm is active.
- The scenarios in which the norm is violated.

To specify the scenarios where a norm is active or violated in, we will state the value of the fluents that will define unequivocally the set of situations that represent those set of states.

The first part of the specification is meant to contain all the possible states in which the norm must be taken in consideration (is active). The second one comprises all the states in which the norm's content is violated. As follows, we present the first part of the specification.

In our proposal a norm N, after doing an action A in situation S, is *active* if and only if it fits in one of these three cases:

i. N was not active before doing A. There is a set of conditions under which A changes the activation state of N from inactive to active. The conditions needed for A to activate N are fulfiled in S.
The *Activation Condition*: Given a certain norm in a situation where the norm is inactive, the range of A is the actions that may change the values of the fluents on which depends the activation state of the norm, in a way that the resultant situation (defined by the resultant value of the fluents) could belong to the scenarios in which the norm is active.

ii. N was active before doing A. There is a set of conditions under which A changes the activation state of N from active to inactive. The conditions needed for A to deactivate N are not fulfiled in S.
The *Maintenance Condition*: Given a certain norm in a situation where the norm is active, the range of A is the actions that may change the values of the fluents on which depends the activation state of the norm, in a way that the resultant situation could belong to the scenarios in which the norm is active.

iii. N was active. There is no set of conditions that can make A change the activation state of N from active to deactivated.
The *Non-Termination Condition*: Given a certain norm and a situation where the norm is active, the domain of A is the actions that may not change the values of the fluents in a way that the resultant situation could belong to the scenarios in which the norm is inactive.

If we analyse these three rules we can assure that every state in which the norm is active fits into one and only one of these three rules. By checking a certain situation with a proposed action we can assert the activation state of any norm after that action has been performed in the situation.

The second part of the specification contains the scenarios where a norm is *violated*. In this case, we have decided to make a simpler specification.

In the activation condition specification we had the temporal progression integrated into it by the use of a variable that represented the action just performed (variable A). In this case we will omit that variable and see the specification of the *violation state* based solely on *the situation's fluents*. It is possible to do that without losing expressivity since the temporal progression in our domain is represented as well in the fluents definition (which specification's looks very much like the activation condition specification), otherwise we would lose the concept of timeline.

By deleting that action variable, the specification becomes much simpler as only the fluent that define the states where the norm is violated have to be stated.

In our proposal a norm N is *violated* in situation S if and only if it fits in one of these two cases:

i. N is active in S, N *obliges* to the value of one or more fluents and S does not fulfil all of those obliged fluents.
This rule is intended to cover the violations done upon norms that oblige to something.

ii. N is active in S, N *forbids* the value of one or more fluents and S fulfils one of those forbidden fluent.
This rule is intended to cover the violations done upon norms that forbid something.

With those two rules we cover the possible violations that can come upon a norm, as norms that allow something cannot be violated. Once having the two parts of the specification of each norm, we can implement them to see how they work once applied to a real life domain.

4 Normative Implementation based on Situation Calculus

In the previous section we have proposed a specification for norms working on dynamic domains under the approach of Situation Calculus and a norm's lifecycle such as active, inactive and violated. Now we will see how this specification can specify real laws on standard Prolog. Specifically, our focus is on the temporal progression aspects involved in the activation state part of the specification. The implementation of the violation state is simpler and we will omit it here; however, the

interested reader can find a Prolog prototype of our normative knowledge base in http://www.lsi.upc.edu/~jcnieves/software/NormativeKnowledge-PAAMS-2010.pl

We are going to consider the Decree 130/2003 of the *Catalan Water Agency*. Remember, that this decree was motivated in §2. As an example, we will study article 8.2 related to an industrial agent of our multiagent system:

> If industrial spills contain limited substances they must respect the established limitations (thresholds).

Following the analysis explained in the previous section, we know that:

- Norm 8.2 is an obliging norm.
- The situations in which the norm is active are those in which an industrial agent makes a spill containing limited substances. The situations in which the norm is violated are those where the substance limitations established are not respected.
- The variables the norm 8.2 applies to are an agent *IdAgent* and a spill *IdSpill*.
- The actions that may change the activation state of the norm are:
  - `set_agent_type`: change the type (domestic, industrial,...) of an agent.
  - `add_substance`: add a certain amount of a substance to a spill.
  - `del_substance`: delete a certain amount of substance from a spill.
  - `del_total_substance`: delete all the contained substance from a spill.
  - `make_spill`: start the execution of a spill by an agent
  - `cancel_spill`: cancel the execution of a spill by an agent
- The preconditions of each action will be defined in a predicate named $poss(A,S)$, following the situation calculus syntax, where A is an action and S is a situation.

The resultant Prolog implementation of applying that information to the specification schema given before for the activation state of a norm is as following: Norm 8.2 is active for agent IdAgent's spill IdSpill after doing action A in situation S
Actions that may activate the norm when the conditions are set

```
holds(norm(82,IdAgent,IdSpill),do(A,S)) :-
A=set_agent_type(IdAgent,domestic),
holds(spill_contains_limited_substance(IdSpill,Substance),S),
not holds(norm(82,IdAgent, IdSpill), S), poss(A, S).

holds(norm(82,IdAgent,IdSpill),do(A,S)) :-
A=add_substance(IdSpill,Substance,Quantity),
holds(agent_type(IdAgent,domestic),S),
holds(agent_spill(IdAgent,IdSpill),S),
holds(substance_limitation(Substance,Limit),S),Limit>0,Quantity>0,
not holds(norm(82,IdAgent,IdSpill),S),poss(A,S).

holds(norm(82,IdAgent,IdSpill),do(A,S)) :-
A=make_spill(IdSpill,IdAgent), holds(agent_type(IdAgent,domestic),S),
holds(spill_contains_limited_substance(IdSpill,Substance),S),
not holds(norm(82,IdAgent,IdSpill),S),poss(A,S).
```

Actions that may terminate it when the conditions are not set

```
holds(norm(82,IdAgent,IdSpill),do(A,S)) :-
holds(norm(82,IdAgent,IdSpill),S),
A=set_agent_type(IdAgent,domestic), poss(A,S).

holds(norm(82,IdAgent,IdSpill),do(A,S)) :-
holds(norm(82,IdAgent,IdSpill),S),
A=del_substance(IdSpill,Substance,Quantity),
holds(spill_contains_limited_substance(IdSpill,Substance2),S),
Substance!=Substance2,poss(A,S).

holds(norm(82,IdAgent,IdSpill),do(A,S)) :-
holds(norm(82,IdAgent,IdSpill),S),
A=del_substance(IdSpill,Substance,QD),
holds(substance_spill_quantity(IdSpill,Substance,QPre),S),
QPre>QD,poss(A,S).

holds(norm(82,IdAgent,IdSpill),do(A,S)) :-
holds(norm(82,IdAgent,IdSpill),S),
A=del_total_substance(IdSpill,Substance),
holds(spill_contains_limited_substance(IdSpill,Substance2),S),
Substance!=Substance2,poss(A,S).
```

Actions that may not terminate the norm

```
holds(norm(82,IdAgent,IdSpill),do(A,S)) :-
holds(norm(82,IdAgent,IdSpill),S),
not A = set_agent_type(IdAgent,Type),
not A=del_substance(IdSpill,Sub,Q),
not A = del_total_substance(IdSpill,Substance),
not A = delete_agent(IdAgent),
not A = cancel_spill(IdSpill,IdAgent),poss(A,S).
```

This resultant implementation, as justified before, fully represents, in *a computable way*, a norm and all the states it may be into: activated, deactivated, violated and respected. When implementing agents that have to interact with a legal framework, we have now a way to integrate the understanding of laws contents and conditions into them.

Now that we have a working implementation of norms, as well as the domain's actions and fluents, we can test the performance of our code by asking Prolog about the norms following the next syntax: to know which norms are applicable to an agent *idAgent* after performing an action *action1* in a given initial state *ini*:

```
holds(norm(X,idAgent),do(action1,ini)).
```

That query will return the complete list of norms active in the state generated after executing *action1* in the scenario *ini*. To know which norms are applicable to an agent's (*idAgent*) spill *idSpill* after performing an action *action2* in a given initial state *ini*:

```
holds(norm(X,idAgent,idSpill),do(action2,ini)).
```

In situation calculus a state is defined as the result of executing an action in another state, therefore we can nest a list of actions to check a norm's state:

```
holds(norm(X,idAgent),do(action2,do(action1,ini))).
```

That query will return the list of norms active for agent *idAgent* after performing *action1* and later *action2*.

It is important to notice that we assume perfect knowledge of our world, which means that the initial state's (on which posterior actions will be performed) fluents are fully asserted.

5 Conclusions

Since norms in real world usually are defined in an abstract level [15], the modeling process of real norms is not straightforward. Some authors has already pointed out that the instantiation of norms in a context domain [15] helps in the process of representing norms in a normative knowledge.

In order to capture the scope of a norm in a context domain, one can fix the observable items which can affect the livecycle of a norm. In particular the representation of these items in terms of fluents/predicates can help to infer the state of a norm. Since the state of a norm will be affected according to the changes of the observable items, one can monitoring the livecycle of a norm in parallel to the changes of the observable items which affect a norm. For exploring the given idea, we have considered situations calculus. Observe that a context domain can be clearly delimited by a set of fluents (a situation) and this was one of main motivations of using situations calculus. As a running example, we have analyzed the Catalan Decree 130/2003 of the *Catalan Water Agency* as a realistic example for managing urban wastewater systems.

In order to incorporate normative knowledge in a situation calculus specification, we proposed to split the specification of norms in two parts: 1.- the scenarios in which a norm is active and 2 .- the scenarios in which a norm is violated. The first part of the specification is meant to contain all the possible states in which the norm must be taken in consideration (is active). The second one comprises all the states in which the norm's content is violated. Since the norms are represented in terms of the fluents of the given domain, the proposed specification represents a natural extension of a situation calculus specification.

In the literature, we can find different approaches for performing normative monitoring [7, 6, 16]. Possibly the approach presented in [7] is close related to the approach presented in this paper. In [7], the authors performs normative monitoring by considering even calculus; however, their approach does not consider a livecycle of a norm.

Some issues for our future work are: 1.- the consideration of a livecycle of actions: by the moment we have assumed actions as an atomic event; hence, this assumption has its limitations for capturing temporal aspects as deadlines. 2.- the consideration of conflicts between norms: for this we are exploring the definition of a partial order between norms.

Acknowledgements

This work has been partially supported by the FP7 European project ALIVE IST-215890. The views expressed in this paper are not necessarily those of the ALIVE consortium.

References

1. C. C. Albrecht, D. D. Dean, and J. V. Hansen. Using situation calculus for e-business agents. *Expert Systems with Applications*, 24:391–397, 2003.
2. H. Aldewereld. *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*. PhD thesis, Utrecht University, 2007.
3. M. Aulinas. *Management of industrial wastewater discharges through agents' argumentation*. PhD thesis, University of Girona, October 2009.
4. CEC. Directive 96/61/EC of 24 September 1996 concerning integrated pollution prevention and control. *Official Journal L*, 257(10):10, 1996.
5. R. Demolombe. From belief change to obligation change in the situation calculus. In *ECAI*, pages 991–992, 2004.
6. R. Demolombe and P. P. Parra. Integrating state constraints and obligations in situation calculus. In *LA-NMR*, 2006.
7. D. Kaponis and J. Pitt. Dynamic specifications in norm-governed open computational societies. In *Engineering Societies in the Agents World VII*, volume 4457 of *Lecture Notes in Computer Science*, pages 265–283. Springer Berlin / Heidelberg, 2007.
8. L. Lesperance, H. Levesque, and R. Reiter. *Foundations and theories of rational agents*, chapter A Situation Calculus approach to modeling and programming agents, pages ?–? 1999.
9. J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. reprinted in McC90.
10. J.-J. C. Meyer and R. J. Wieringa, editors. *Deontic Logic in Computer Science: Normative System Specification*. Wiley, 1993.
11. E. C. Ministry. Decree 130/2003, Reglament dels serveis públics de sanejament. *DOGC*, 3894:11143–11158, 2003.
12. N. Oren, S. Panagiotidi, J. Vázquez-Salceda, S. Modgil, M. Luck, and S. Miles. Towards a formalisation of electronic contracting environments. *Coordination, Organization, Institutions and Norms in Agent Systems, the International Workshop at AAAI 2008, pages 61-68, Chicago, Illinois, USA*, 2008.
13. S. Panagiotidi, J. C. Nieves, and J. Vázquez-Salceda. A framework to model norm dynamics in answer set programming. In *Proceedings of FAMAS' 2009 (Multi-Agent Logics, Languages, and Organisations Federated Workshops)*, volume 494 of *CEUR WS Proceedings*, pages ?–?, 2009.
14. R. Reiter. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, chapter The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression, pages 359–380. Academic Press Professional, Inc, 1991.
15. J. Vázquez-Salceda. *The Role Of Norms And Electronic Institutions In Multi-Agent Systems Applied To Complex Domains The Harmonia Framework*. Artificial intelligence, LSI (Universitat Politècnica de Catalunya), Barcelona, Spain, 2003.
16. J. Vázquez-Salceda, H. Aldewereld, D. Grossi, and F. Dignum. From human regulations to regulated software agents' behavior. *Artif. Intell. Law*, 16(1):73–87, 2008.