Brigham Young University
BYU ScholarsArchive

International Congress on Environmental Modelling and Software

5th International Congress on Environmental Modelling and Software - Ottawa, Ontario, Canada - July 2010

Jul 1st, 12:00 AM

# Reasoning about Actions for the Management of Urban Wastewater Systems using a Causal Logic

Dario Garcia-Gasull

Juan Carlos Nieves

Ulises Cortés

Follow this and additional works at: https://scholarsarchive.byu.edu/iemssconference

Garcia-Gasull, Dario; Nieves, Juan Carlos; and Cortés, Ulises, "Reasoning about Actions for the Management of Urban Wastewater Systems using a Causal Logic" (2010). *International Congress on Environmental Modelling and Software*. 249.
https://scholarsarchive.byu.edu/iemssconference/2010/all/249

This Event is brought to you for free and open access by the Civil and Environmental Engineering at BYU ScholarsArchive. It has been accepted for inclusion in International Congress on Environmental Modelling and Software by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

International Environmental Modelling and Software Society (iEMSs)
2010 International Congress on Environmental Modelling and Software
Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada
David A. Swayne, Wanhong Yang, A. A. Voinov, A. Rizzoli, T. Filatova (Eds.)
http://www.iemss.org/iemss2010/index.php?n=Main.Proceedings

# Reasoning about Actions for the Management of Urban Wastewater Systems using a Causal Logic

**Dario Garcia-Gasulla,** [a] **Juan Carlos Nieves**[a] **and Ulises Cortés**[a]

[a]*Knowledge Engineering and Machine Learning Group, Universitat Politècnica de Catalunya, C/Jordi Girona 1-3, Omega 201, Barcelona, Spain ({dariog, jcnieves, ia}@lsi.upc.edu)*

**Abstract:** The management of Urban Wastewater Systems (UWS) is a critical and difficult process. Handling polluted water in UWS requires planning and decision making regarding sectors such as environmental, energetic, industrial, etc. In order to represent all the significant concepts, a powerful and flexible formalism is needed. Nowadays, there are solid approaches based on nonmonotonic reasoning which can capture dynamic domains such UWS. In this paper, we present an implementation of a real UWS in the action language Causal Calculator (CCalc-C+) which displays all the singularities of such domain and within which complex queries can be performed. The nonmonotonic causal theory, upon which is based CCalc, allows us to represent some characteristics of UWS such as nondeterministic actions and indirect effects of actions. We will capture the situations and the transitions of the UWS domain and represent them in the form of a finite state machine. CCalc allows us to perform queries regarding; the effects of actions, action planning to reach a certain state and path finding. The resultant work can be used in actual UWS to aid at the decision making process, simplifying the reasoning about actions and allowing a close representation of the real state of UWS.

***Keywords:*** Wastewater management, Reasoning about actions, Nonmonotonic causal logic.

## 1 INTRODUCTION

Decision-making and planning in environmental context is a delicate and crucial task. The amount of data that defines the state of entities such as Urban Wastewater Systems (UWS) can be extremely large and technical, and the repercussions of the decisions taken huge. To aid in the process of planning and taking decisions it is necessary to create a formal representation of all the elements involved, a model. In order to help modelize the situation of a realistic dynamic domain such as the one of the UWS, a special formalism is needed.
The inherent characteristics of UWS, such as nondeterministic actions, indirect effects and concurrency of events, requires a flexible and powerful logic formalism. In this context, nonmonotonic causal theories introduced by McCain and Turner [1997] can be useful and effective, as proved by Akman et al. [2004] and Lifschitz et al. [2000]. By using the implementation of this approach to action's effects description called the Causal Calculator (CCalc) we can represent and query real-life domains.

Following the syntax of CCalc presented by Giunchiglia et al. [2004] we propose an implementation of a Wastewater Treatment Plant (WWTP) and some of its internal key elements. We will represent the situations of a WWTP in the form of an automaton which will recreate the states a WWTP may be in. Traveling through that automaton will be the same as traveling through the states of the domain, following the happening events and the effects those actions may cause on the world. Those elements will be implemented by the use of fluents and actions. Once having the WWTP realistically represented we will then be able to propose a series of queries to be performed on that implementation which can aid at the daily management of the WWTP. Concretely,

we will see three different types of queries: a) queries to simulate future scenarios, b) queries to learn how we got to the current state and c) queries to know how to optimally get to a certain situation.

The rest of the paper is divided as follows: In §2, a brief introduction to the background logic and the syntax of CCalc is presented in order to facilitate the understanding of the solution proposed. In §3 we present and modelize a particular case. A first generic analysis of it is done which will serve as preamble of §4. In §4 the final implementation is described, its main elements are explained and an example of a performed query and its interpretation is discussed. In §5 we present our conclusions and future work.

## 2 BACKGROUND

WWTPs and their activities are very complex. The performed actions taken may not always have the same result, there are important side effects and time is a key and active element in every process, as WWTPs deal mostly with pollutive wastewaters. To faithfully represent such a domain, a logic formalism which can handle concurrency, nondeterminism and temporariness is necessary. Nonmonotonic causal logic, by assuming a particular version of the principle of universal causation (PUC), $Everything\ that\ is\ true\ has\ a\ cause$, uses a simple mathematical syntax, has great expressiveness and is able to represent all those necessary concepts. The particularity of the PUC used is that in it, a fact can be its own cause. CCalc is an implementation of the aforesaid approach and defines the domain mainly by the use of fluents and actions.
Fluents tell us the truth value of a fact in a given moment and altogether define the current situation. Fluents may have inertial functions to define their behavior, so that, once their value is established, and as long as no action changes it, their value follows a certain pattern.
On the other hand, actions define the events happening in our domain that change our world, that is, the value of fluents. Actions can be exogenous, i.e. they may be triggered without a cause inside our domain, e.g. by external human interaction.

This kind of approach to formalize biological domains (i.e. by the use of actions and fluents) has been used before, as proposed by Dworschak et al. [2008], Nieves et al. [2009] and Nieves et al. [2010], but using different approaches as $Situation\ Calculus$, $Event\ Calculus$ and action languages such as $A$ and $C_{TAID}$. While all of them are based on the *fluents and actions* paradigm in order to represent dynamic domains, $Situation\ Calculus$, $Event\ Calculus$ and $A$ do not have some interesting and complex features such as nondeterminism. This features will be very useful when trying to put the solution to work in a real application. $C_{TAID}$ does allow most of CCalc's features, but the semantic simplicity provided by the causal logic defined behind $C+$ is lost in $C_{TAID}$, which is an extended version of actions languages *A* and *C* in the direction of $C+$.

CCalc most distinguishing element in front of these other methods are the causal rules (except $C_{TAID}$, which contains an adapted version called *allowance rules*). Causal rules can define the effects and preconditions of every action, the relationships between fluents and even the domain constraints, stating which situations are not possible and in which states certain actions cannot happen. In CCalc, and following the PUC, everything that does not have a cause is false. Therefore causal laws must state everything that is caused and how.
In order to understand the technical issues of the rest of this paper, a simplified introduction to some of the more usual causal rules syntax of CCalc will be presented below. A complete and deep explanation of nonmonotonic causal theory and CCalc can be found in Giunchiglia et al. [2004]. The causal laws of CCalc work with formulae. Formulae have a domain which contains all its possible values and can be of two main kinds: fluent formulae, representing fluents of the domain, and action formulae, representing actions performed on the domain.

In CCalc a boolean formula $F$ with the domain $\{TRUE, FALSE\}$:

- Is caused when another formula $G$ is executed if: $caused\ F\ if\ G$
  If $F$ does not require of another formula ($G = TRUE$): $caused\ F$

- Is caused after another formula $G$ is executed if: $caused\ F\ after\ G$
- Cannot be caused after another formula $G$ is executed if: $constraint\ \neg F\ after\ G$
  If $F$ can never be caused: $constraint\ \neg F$
- Is a fluent constant which cannot change through time: $rigid\ F$
- Is a fluent constant which keeps its value unless a causal law changes it: $inertial\ F$
  If this is not specified for a certain fluent, its value for every state will have to be explicitly stated or considered unknown.
- Is true unless a specific causal law states otherwise: $default\ F$
- Is true unless a specific causal law states otherwise after another formula $G$ is executed if: $G\ may\ cause\ F$
- Is an action constant which can be true or false without direct interaction within the domain: $exogenous\ F$
  If an action is not defined as exogenous, in order to execute it we will need a causal rule to justify it (because of the PUC). An action can be defined as exogenous and have causal rules controlling its behavior in certain situations, which in those cases will override the exogenous, by default, behavior.

This generic and simplified view of some of the causal laws of CCalc is enough to understand the expressiveness power and utility of CCalc applied to the domain of UWS. These are the basic ones and will be enough to represent our use case. However, CCalc provides some more kind of laws.

## 3 USE CASE

In our attempt to formalize the behavior of a WWTP, and due to the lack of expert knowledge on the matter, we will work with a simplified interpretation of a WWTP similar to the one presented by Gimeno et al. [1997] and by Nieves et al. [2009]. The most important elements will be represented, and the effects of actions may not be completely accurate. The variables involved will be quantized to simplify the solution and for efficiency reasons. Due to CCalc's way of grounding variables and calculating arithmetic expressions, the use of large numeric intervals in the domain of variables is not advisable. That point is further explained in §5.
In our use case most of the problems of formalizing dynamic domains will be present (such as nondeterminism and concurrency), and modifying it so it becomes a faithful representation of a complete WWTP (through expert knowledge) should not be complicated. It is as follows.

In order to achieve the required outflow water quality, the WWTP processing of water includes four main stages through which all arriving water must pass. On the first one, the entry, the water is accepted or rejected into the WWTP. From the entry, the water reaches the second stage, the settler, where the solid substances are separated from the rest. The formed sludge from those substances can be removed by purging the settler. After the settler, the water arrives at the bioreactor which contains micro-organisms that help clean the water. These micro-organism must be kept alive by regulating the oxygen level on the bioreactor. That can be achieved by oxygenating the water of the bioreactor. After the bioreactor the water is finally send to the last stage, what we will call the exit. In the exit the solid substances are separated from the water again (as in the settler). After that, the clean water can be expelled to the environment or, if it is detected that the water is not clean enough, sent back to the settler to start the whole process again.

From the previously explained use case, we have extracted a number of fluents and actions which represent the whole domain. We will show here a simplified version of them, as showing them in CCalc syntax would only make its understanding more difficult.

Fluents:

| | |
|---|---|
| $plant\_load$: Total water load of the WWTP. | $plant\_situation$: Current situation of the plant (normal, emergency,...) |
| $entry\_dirt$: Dirt level of the water entering the WWTP. | $entry\_load$: Water load at the entrance of the WWTP. |
| $settler\_load$: Water load of the settler. | $settler\_sludge$: Amount of sludge in the settler. |
| $settler\_dirt$: Dirt level of the water in the settler. | $bioreactor\_load$: Water load of the bioreactor. |
| $bioreactor\_oxygen$: Oxygen level of the bioreactor. | $bioreactor\_dirt$: Dirt level of the water in the bioreactor. |
| $exit\_load$: Water load of the exit. | $exit\_sludge$: Amount of sludge in the exit. |
| $exit\_dirt$: Dirt level of the water in the exit. | $environment$: Current situation of the environment (normal, emergency,...). |

Actions:

| | |
|---|---|
| $water\_arrival$: Water arrives to the WWTP. | $entry\_accept(X)$: Arrived water is accepted into the WWTP. |
| $entry\_bypass$: Arriving water is rejected into the WWTP. | $settler\_purge$: Purge the settler of sludge. |
| $settler\_pass$: Pass water from the settler to the bioreactor. | $bioreactor\_oxygenate$: Increase the oxygen level of the water in the bioreactor. |
| $bioreactor\_pass$: Pass water from the bioreactor to the exit. | $exit\_purge$: Purge the exit of sludge. |
| $exit\_discharge$: Return water to the environment. | $exit\_recharge$: Reload water to the settler. |

Once we have those fluents and actions specified in CCalc we need to define the set of causal rules that will dictate their behavior. Concretely, we will implement rules that determine:

- The concurrent execution of actions: Which actions can and which cannot be executed concurrently and under which circumstances. By default we will set the scenario as concurrent, and the incompatible actions will be stated explicitly.
- The water flow through the WWTP stages: How certain actions affect the water load of the stages and the restrictions of these actions.
- The pollution management: Dictate the pollution level, how it changes with the income of new water and the reduction achieved by the actions available in the four treatment phases.
- The state of the WWTP as a whole: How the plant situation depends on its internal stages situation.
- The state of the environment: Set the effects of the WWTP decisions on the environment situation.

## 4 IMPLEMENTATION

In order to implement correctly what was analyzed in the previous section we need to understand how CCalc works. First of all, it is required to study the fluents and actions and obtain a list of the different data types we will have to work with. In order to simplify our solution we have decided to use a same categories to express similar concepts. The amount of water, dirt level, sludge amount and oxygen level, will all have the domain: {*low, medium, high*}. For the plant situation we have defined two of the main problems of WWTP ({*bulking, foaming*}) and two generic states ({*normal, alarm*}). Two possible situations will be defined for the environment, normal and emergency, in case a polluted discharge is made.

```
:- sorts
plant_situation; state; environment.
:- objects
normal, bulking, foaming, alarm ::  plant_situation;
low, medium, high ::  state;
normal, emergency ::  environment.
```

Once having the types of data and the objects within them we need to study the kind of fluents and actions we will use. In CCalc the most common fluent type is `inertialFluent`. This kind of fluent can have any possible initial value (within its domain) and if no action states otherwise (through a causal rule) it will keep its value along the timeline (hence the `inertial`). Most of our fluents will be of this kind, as they will be specified by actions and keep their value by through time by default. We will define as well two statically determined fluents. This kind of fluent is not affected by any action directly and its value depends solely on other fluent values. This will be the case of `plant_load` and `plant_situation`, which depend on the value of the internal stage fluents. As actions change these fluents, the statically determined fluents will change as well. That can be seen as indirect effects of those actions.

```
plant_load :: simpleFluent(state);
plant_situation :: simpleFluent(plant_situation);
entry_dirt :: inertialFluent(state);
entry_load :: inertialFluent;
settler_load :: inertialFluent(state);
settler_sludge :: inertialFluent(state);
settler_dirt :: inertialFluent(state);
bioreactor_load :: inertialFluent(state);
bioreactor_oxygen :: inertialFluent(state);
bioreactor_dirt :: inertialFluent(state);
exit_load :: inertialFluent(state);
exit_sludge :: inertialFluent(state);
exit_dirt :: inertialFluent(state);
environment :: inertialFluent(environment);
```

Actions in CCalc are usually defined as `exogenousAction`. This definition states that the action is not triggered within the domain, that is, for any given moment the action may happen without any explicit reason inside our domain. All of our actions will be of that kind.

```
water_arrival :: exogenousAction;
entry_accept :: exogenousAction;
entry_bypass :: exogenousAction;
settler_purge :: exogenousAction;
settler_pass :: exogenousAction;
bioreactor_oxygenate :: exogenousAction;
bioreactor_pass :: exogenousAction;
exit_purge :: exogenousAction;
exit_discharge :: exogenousAction;
exit_recharge :: exogenousAction.
```

Regarding causal rules, and considering the implicit rules generated by the definitions of fluents as `inertial` (for example, if `settler_load` is high, it will keep being high as long as no explicit causal rule changes it) and actions as `exogenous` (for example, the action `water_arrival` can be executed at any moment, without warning or cause), we will only have to implement causal rules for the next purposes:

- To state the effects of an action on a fluent. This rules may have preconditions for the effect to take place. For example:
  *When a discharge is done, if the water load at the exit was medium, the resultant water load will be low.*
  `exit_discharge causes exit_load=low if exit_load=medium`
- To state the nondeterministic effects of an action on a fluent. This rules may as well have preconditions to be executed. For example:
  *Accepting very dirty water into the settler may increase the sludge level of it to high.*
  `entry_accept may cause settler_sludge=high if entry_dirt=high`

The `may cause` rules generate as many equiprobable states as necessary to represent all the nondeterministic effects of the rule. If a deterministic causal rule specifies on the same fact, the nondeterministic effects are discarded and only the deterministic effect is taken into account.

- To regulate the concurrent execution of actions. CCalc allows concurrency of actions unless otherwise stated (several actions happening concurrently in the same time step). By default we allow concurrency, and to control the exceptions we will define non executability rules. These rules will not allow the current execution of two actions that in our domain cannot happen at the same time. In the same way, they will not allow the execution of an action in a situation where the action cannot happen. For example:
  *The bioreactor cannot pass water to the exit if it is oxygenating.*
  `nonexecutable bioreactor_pass if bioreactor_oxygenate`
  And:
  *The WWTP cannot accept more water if the first stage water load is high.*
  `nonexecutable entry_accept if settler_load=high`
- To define the static relationship between two fluents. We will use these fluent static rules to link the value of a set of fluents to the value of another set. For example:
  *The WWTP has a foaming problem if the bioreactor level is low and the dirt in the bioreactor water is high.*
  `caused plant_situation=foaming if bioreactor_oxygen=low, bioreactor_dirt=high`

Due to the lack of space we will not display the whole code here, but we consider that the previous examples are sufficient representative. The complete source can be found in the URL: `http://www.lsi.upc.edu/~dariog/soft/wwtp.pl`
Finally, once we have our domain implemented, we can define queries on it. There are three main kinds of queries in CCalc:

- Queries to study the effect of one or more events on the domain. For example: Which will be the future results, for the environment and the WWTP, of accepting very dirty water into the WWTP?
- Queries to find out the possible paths which will lead to a given situation. For example: Which are the possible events that lead to the current situation in which the WWTP had a bulking problem?
- Planning queries to find the shortest list of actions needed to achieve a certain objective. For example: Which set of actions must be executed in order to achieve a normal situation in the WWTP without damaging the river quality?

Next, we will see the execution of a query and the results of it. We start in a situation (defined by the time stamp `0:`) where both the environment and the plant are in a normal situation, where we have dirt and load in both bioreactor and exit, and where a new load of fairly dirty water has arrived. In our goal state (defined in `maxstep` :) both the environment and the treatment plant are in normal situation and the dirt level of the bioreactor and the exit is low. We want to reach that state in a maximum of three time steps (defined by `maxstep` :: 3). We define as well a new arrival of water after one step and a discharge into the river after two steps.

```
:- query
label::1;
maxstep::3;
maxstep:  environment=normal, plant_situation=normal,
bioreactor_dirt=low, exit_dirt=low;
0:  environment=normal; 0:  plant_situation=normal;
0:  entry_dirt=medium; 0:  settler_load=low;
0:  settler_sludge=medium; 0:  settler_dirt=low;
0:  bioreactor_load=high; 0:  bioreactor_oxygen=high;
0:  bioreactor_dirt=high; 0:  exit_load=high;
0:  exit_sludge=medium; 0:  exit_dirt=high;
```

```
0:  entry_load;
1:  water_arrival;
2:  entry_dirt=medium;
2:  exit_discharge.
```

This query returns a list of 49 possible sequences of events which fulfill all the requirements set (as we allowed concurrency in the scenario, several actions are performed at each time step). If we analyze carefully these possible worlds we will realize that such a large number of solutions is the effect of the nondeterminism of certain actions. In fact there is only two possible solutions. From the sequences obtained, all of them contained the same actions at time 0: `entry_accept` and `exit_purge`. After the execution of these two actions, in the second time step, the actions performed must be: `settler_purge`, `exit_purge` and `water_arrival` (specified by us in the query). Afterwards, in the third and last step: `entry_bypass` and `exit_discharge` (specified by us in the query). The action `settler_purge` done in the second step appears sometimes in the third step, indicating that the action must be performed but it can be done in the second or the third step, which represents the two possible solutions.
After analyzing both solutions, we have the list of sequential actions that must be performed. In this scenario, and to achieve the objectives set, we would recommend purging the exit two times to decrease the sludge amount to a minimum and at the end make an entry bypass. Otherwise one of the requirements set in the query would be violated.

## 5 CONCLUSIONS

After performing several queries, analyzing the results obtained and how these results could contribute to the management of a WWTP, we reached the conclusion that CCalc is a good choice when formalizing and querying complex dynamic domains, as the one of the UWS. CCalc's simple and common decomposition of the domain in actions and fluents, together with the expressiveness power of the nonmonotonic causal logic, provides a relatively simple and realistic representation of the internal work of a WWTP.
The simple mathematical syntax of CCalc facilitated a plain implementation of nondeterministic actions, indirect effects of actions and inertial behavior of fluents. As expected, the incorporation of those particular elements generated interesting results in the formalization and later testing of the model. Implementation of indirect effects was easily resolved by the use of statically determined fluents, which provided the means to rapidly check complex properties. The inertial declaration of fluents behavior, one of the most important properties of CCalc, is a very elegant solution to one of the biggest problems of dynamic domains modeling: The frame problem, which was defined by McCarthy and Hayes [1987].
Probably the most enriching feature regarding the final performance of the model is the nondeterminism of actions. Defining and querying a domain which contains this property offered interesting consequences. To start with, nondeterministic actions caused a notable increase of the solutions obtained to most queries, as seen in §3. The apparent repetition of solutions is a necessity when trying to obtain all the possible scenarios which contains uncertain effects of actions. Only a user provided refining of the query parameters, stating unambiguously the effects of nondeterministic actions, could produce a reduced and yet valid set of answers. This feature could bring along new possibilities for CCalc to aid at the management of WWTPs, as the agents involved could perform different queries assuming the best or the worst possible scenario, based on a basic unbounded query.

The most significant limitation of CCalc we encountered was the way it deals with numeric elements. As CCalc grounds each rule with all possible instances of the numeric variables defined and calculates arithmetic expressions logically, the use of large numeric ranges may affect its efficiency notoriously. To deal with this problem we chose to quantize all the potentially numerical variables and used comparison operators instead of arithmetic, which entailed a simplified but very efficient version of the domain.

In conclusion, our tests and experience with CCalc proved it to be a practical and powerful tool

for implementing dynamic domains. It can be very useful in the task of aidding at the decision making, planning and path finding process of realistic scenarios as long as its peculiarities are understood and taken into account.

### 5.1 Future work

The large set of answers given to most queries involving nondeterministic actions arouses an interesting possibility, the post-process of the solution set. A software in charge of analyzing and displaying the results obtained to uncertain queries could get to be a very useful and elaborated tool, taking into account the number of possible outcomes, their historic likelihood and their probability to happen. To apply the latter idea, modifications on the semantic of CCalc would be necessary, as CCalc's nondeterministic causal rules do not allow probability setting.
Considering CCalc use of mathematical elements, and in order to obtain a more realistic domain, the fluents and causal rules could be quantified to a certain degree, testing this way CCalc's balance between expressiveness and efficiency.
Finally, it would be interesting to see, knowing that the domain of the UWS is strictly ruled by several normative sets, the implementation of norms in CCalc and its inclusion and interaction with the rest of the domain model.

#### ACKNOWLEDGMENTS

The authors wish to acknowledge Selim Erdoğan for his assistance with CCalc and Miquel Sànchez-Marrè for his help in the process of understanding the world of the WWTPs.

#### REFERENCES

Akman, V., S. T. Erdoğan, J. Lee, V. Lifschitz, and H. Turner. Representing the zoo world and the traffic world in the language of the causal calculator. *Artif. Intell.*, 153(1-2):105–140, 2004.

Dworschak, S., S. Grell, V. J. Nikiforova, T. Schaub, and J. Selbig. Modeling biological networks by action languages via answer set programming. *Constraints*, 13(1-2):21–65, 2008.

Gimeno, J. M., J. Béjar, and J. Lafuente. Providing wastewater treatment plants with predictive knowledge based on transition networks. In *IIS '97: Proceedings of the 1997 IASTED International Conference on Intelligent Information Systems (IIS '97)*, page 355, Washington, DC, USA, 1997. IEEE Computer Society.

Giunchiglia, E., J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artif. Intell.*, 153(1-2):49–104, 2004.

Lifschitz, V., N. McCain, E. Remolia, and A. Tacchella. Getting to the airport: the oldest planning problem in ai. pages 147–165, 2000.

McCain, N. and H. Turner. Causal theories of action and change. In *AAAI/IAAI*, pages 460–465, 1997.

McCarthy, J. and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. pages 26–45, 1987.

Nieves, J. C., M. Aulinas, and U. Cortés. Reasoning about actions for the management of urban wastewater systems: Preliminary report. In *Frontiers in Artificial Intelligence and Applications (CCIA'2009)*, pages 371–380, Amsterdam, The Netherlands, 2009. IOS Press.

Nieves, J. C., D. Garcia-Gasulla, M. Aulinas, and U. Cortés. Using situation calculus for normative agents in urban wastewater systems. In *Accepted in the 7th International Conference on Practical Applications of Agents and Multi-Agent Systems, Advances in Intelligent And Soft Computing, April 26-28*, Salamanca, Spain, 2010. Springer.