Citation for published version:
De Vos, M, Padget, J & Satoh, K 2011, Legal modelling and reasoning using institutions. in New Frontiers in Artificial Intelligence - JSAI-isAI 2010 Workshops, LENLS, JURISIN, AMBN, ISS, Revised Selected Papers. vol. 6797 LNAI, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer, Heidelberg, pp. 129-140, 2nd JSAI International Symposia on Artificial Intelligence, JSAI-isAI 2010, November 18, 2010 - November 19, 2010, Tokyo, Japan, 1/01/11. https://doi.org/10.1007/978-3-642-25655-4_12

DOI:
10.1007/978-3-642-25655-4_12

Publication date:
2011

Document Version
Peer reviewed version

Legal Modelling and Reasoning using Institutions

Marina De Vos[1], Julian Padget[1], and Ken Satoh[2]

[1] Department of Computer Science, University of Bath
{mdv, jap}@cs.bath.ac.uk
[2] National Institute of Informatics
ksatoh@nii.ac.jp

Abstract. To safeguard fairness for all parties involved and proper procedure, actions within a legal context are heavily constrained. Detailed laws determine when actions are permissible and admissible. However, these restrictions do not prevent participants from acting. In this paper we present a methodology to support legal reasoning using institutions—systems that specify the normative behaviour of participants—and a corresponding computational model. We show how it provides a useful separation between the identification of real world actions, if and how they affect the legal model and how consequences *within* the legal model can be specified and verified. Thus, it is possible to define a context, introduce a real-world event and examine how this changes the state of the legal model: hence, the modeller can explore both model adequacy and that of the legal framework from which it is derived, as well as offering a machine-usable legal 'oracle' for software components. We illustrate the use of our framework by modelling contract cancellation under Japanese contract law.

Introduction

The concept of normative frameworks, or institutions, and organizational modelling is gaining acceptance in the multi-agent systems (MAS) as a potential solution to the challenge of governing open systems. Modelling frameworks typically provide a method for the formalization of norms, thus capturing some statement of the designer's intentions for how participants *should* behave: which actions they are allowed to perform and which ones are not permitted. That does not necessarily ensure compliant behaviour: violation detection is generally feasible, as long as agents' actions are somehow observable, but enforcement is a separate issue, which has many implementations on the spectrum between social and legal punishment.

For the last four years, we have been developing a modelling framework and toolset, based on the logic programming language of Answer Set Programming [3], that addresses the governance of multi-agent systems. The aim is to express the range of actions, and constraints, on software agents in a machine processable form. The designer can (exhaustively) explore the reachable states of the model as a means to validate it and in a running system, the model can be used to compare agent behaviour against expectations. It may also function as an oracle service for agents to query at run time.

The preoccupations of just building a system that behaves as desired tends to occlude the precept that agents, the actors, are *situated* and that situation does not include just the physical environment that agents can sense and act upon [12], but should also take account of the (multiple) legal contexts in which such systems may be deployed. An essential aspect of our modelling framework is its support for the distinction—and connection—that the designer can make between real world events and institutional events, following the principle of "counts-as" established by Jones and Sergot [11]. This "counts-as" relation, effective on events, is different from the count-as relations discussed in [6] that operate on facts and are context dependent. Since this institutional interpretation of events is a declarative program, it can be a formal representation of some fragment of a legal code, thus building a connection between agents and a situating legal framework. In this paper we use a Japanese contract law as a case study.

1 Institutions

Formal Model Our formal model has been described in detail in [1], but to make this paper self-contained we provide a brief overview here.

The premise of our model is that events trigger the creation of institutional facts, inspired by Jones and Sergot' [11] account of institutional power and the notion of 'counts-as', to explain the connection between exogenous events and institutional events—this is our *generation* relation. An event may change the institutional state by initiating or terminating fluents—this is our *consequence* relation. The state consists of brute domain facts [9] and institutional facts specifying permission and power, of events and of obligations to perform an action. Thus, given an event and an institutional model state, represented as a set of (institutional) facts, the next state is determined by the transitive closure of the generation and consequence relations. The analogy we make for legal reasoning is that the generation relation models actions of individuals in the real world, creating institutional events in the pertinent legal context, consequently initiating and terminating domain and institutional facts by means of the consequence relation.

The formal institutional model is necessarily more detailed and precise than the sketch above. The essential elements are: (i) events ($\mathcal{E}$), that bring about changes in state, and (ii) fluents ($\mathcal{F}$), that characterise the state at a given instant, where a fluent is a term whose presence in the institutional state indicates it is true, and absence implies falsity. We distinguish two kinds of events: institutional ($\mathcal{E}_{inst}$), that are the events defined by the model and exogenous ($\mathcal{E}_{ex}$), that are outside its scope, but whose occurrence triggers institutional events reflecting the counts-as principle. We partition institutional events into institutional actions ($\mathcal{E}_{act}$) that denote changes in institutional state and violation events ($\mathcal{E}_{viol}$), that signal the occurrence of violations. Violations may be generated explicitly, or through the occurrence of a non-permitted event, or from the failure to fulfil an obligation. We distinguish two kinds of fluents: *institutional* that denote institutional properties of the state such as permissions $\mathcal{P}$, powers $\mathcal{W}$ and obligations $\mathcal{O}$, and *domain* $\mathcal{D}$ that correspond to institutional framework specific properties. The set of all fluents is denoted as $\mathcal{F}$.

The evolution of the state of the framework is achieved through the definition of two relations: (i) the generation relation: that specifies how the occurrence of one (exogenous or institutional) event generates another (institutional) event, subject to the empowerment of the actor. Formally, this can be expressed as $\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{inst}}$, where $\mathcal{X}$ denotes a formula over the (institutional) state and $\mathcal{E}$ an event, whose confluence results in an institutional event, and (ii) the consequence relation, that specifies the initiation and termination of fluents subject to the performance of some action in a state matching some expression, or formally $\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$.

Again, for the sake of context, we summarize the semantics of our framework and cite [1] for an in-depth discussion. The semantics is defined over a sequence, called a trace, of exogenous events. Starting from the initial state, each exogenous event is responsible for a state change, through initiation and termination of fluents, that is achieved by a two-step process: (i) the transitive closure of $\mathcal{G}$ with respect to a given exogenous event determines all the (institutional) events that result from this event, including violations of events that where not permitted and violations events of unfulfilled obligations, while excluding events that are not empowered. (ii) the application of $\mathcal{C}$ to this set of events, identifies all fluents to initiate and terminate with respect to the current state in order to obtain the next state; this also includes the termination of obligations that have been fulfilled or violated. So for each trace, we can obtain a sequence of states that constitutes the model of the institutional framework.

InstAL While the formal framework has its value, from a designer's point of view it is not very convenient and forces technical/mathematical details to intrude into the modelling process. Consequently, we designed a simple domain-specific language for institutional frameworks called Inst*AL* . We give a brief introduction through examples of the language features taken from the case study on Japanese contract law which features later in this paper:

- `institution` *name* declares the name of the institutional framework, such as `institution legal`
- `type` *identifier* declares a type, such as `type Agent`. Type declarations establish a disjoint set of mono-morphic types.
- `exogenous event` *event-name*(*type*$^+$) declares a new physical world event and the types of its parameters, such as `exogenous event sale(Agent, Agent)`
- `inst event` *event-name*(*type*$^+$) declares a new legal world event and the types of its parameters, such as `inst event intSale(Agent, Agent)`.
- `violation event` *event-name*(*type*$^+$) declares a new violation event, such as `violation event contractViolationBuyer(Agent)`
- `fluent` *fluent-name*(*type*$^+$) declares a new institutional fact—that is, an object that can be an element of the institutional state, such as `fluent contract(Agent, Agent)`
- *event-name* `generates` *legal-world-event*$^+$ [*if state-condition*] adds a new pair to the generation relation with domain event (physical or legal) and range legal world event, subject to an optional condition on the state. For example: `sale(Seller, Buyer) generates intSale(Seller, Buyer) if hasGood(Seller), hasMoney(Buyer)`, where the condition is the presence of the fluents `hasGood` and `hasMoney` with the corresponding Seller and Buyer (these variables are unified) in the institutional state.
- *event-name* `initiates` *legal-world-fluent*$^+$ [*state-condition*] adds a new pair to the consequence (addition) relation, with event (physical or legal) and fluent. Thus,

`sale(Seller, Buyer) initiates contract(Seller, Buyer)` adds the corresponding fluent to the institutional state.

– *event-name* `terminates` *legal-world-fluent*$^+$ [*condition*] adds a new pair to the consequence (deletion) relation, with event (physical or legal) and fluent. Thus, `transfer(Seller) terminates hasGood(Seller)` deletes the corresponding fluent from the institutional state.
– `perm`(*event*) is a special fluent whose presence indicates that the event is permitted, such as `perm(transfer(Seller))`, and is typically the subject of an `initiates` or `terminates` rule.
– `pow`(*event*) is a special fluent whose presence indicates that the event is empowered (has an effect), such as `pow(transfer(Seller))`, and is typically the subject of an `initiates` or `terminates` rule.
– `obl`(*event*, *event*, *event*) is a special fluent whose presence indicates the existence of an obligation, such as `obl(transfer(Seller), handOverDeadline, contractViolationSeller(Seller))`, indicating that the first event needs to occur before the second. If this is not the case, the third event, normally a violation event will take place. In either case, the obligation is terminated. Typically the subject of an `initiates` rule.

We realize the above specification by translation to a (non-monotonic) logic programming language call Answer Set Programming (ASP) [3]. This translation includes code to take care of inertia of fluents and removal of satisfied or violated obligations. Again the full details can be found in [1], including a proof of the soundness and completeness of the translation. The result of the translation is a computational model that can generate sequences of all possible states of the institutional framework (these are the answer sets of program), given some initial conditions. The development, execution and visualization of institutions is supported by the IDE Inst*Suite* [8].

Methodology Our institution terminology refers to *exogenous* and *institutional* events. From the perspective of a legal domain, the first represent the actions of the participants: e.g. signing a contract, while institutional events are a mechanism for the legal interpretation of these actions. For example, a signature of a minor is not recognised or stating that one cancels a contract does not automatically mean that the contract has indeed been cancelled; certain conditions must be fulfilled for a contract to be cancelled. Institutions make explicit this separation of concerns and allow reasoning on both levels.

In our model, each exogenous event of concern has a corresponding institutional event, that functions as a gatekeeper to the legal model. If the actor of an exogenous event does not have the necessary credentials or certain conditions are not met, the institutional event does not occur and nothing changes within the model, so the exogenous event is of no consequence. The credentials are modelled by assigning institutional power to the the gatekeeper events. Without institutional power, these events are not triggered. Permission of exogenous events and gatekeeper events is determined by the legal rules that are modelled.

Each institutional event can generate further institutional events to allow for the specification of special or alternative cases, e.g. buyers and sellers have different obligations when a contract is established.

Case:
Contract:

- X had a contract to buy a piece of jewellery from Y for 1 million yen.
- X was supposed to pay 1 million yen on Sep 30 2009 at Y's residence and
- Y was supposed to give the jewellery to Y on the same day.

2007 Sep 30:

- X gave 1 million yen to Y, but Y did **not** give the jewellery to X, so X demanded to Y to give the jewellery J by Oct 14

2007 Oct 14:

- Y did **not** give the jewellery to X

2007 Oct 15:

- X cancels the contract

Question: Is the cancellation of the contract by X valid?

Fig. 1. Contract Cancellation Scenario

Obligations are used to express when future events have to take place, e.g. the contract requires that the goods are transferred before a certain date. To make the model as abstract as possible, dates are not encoded in the model instead we use exogenous events that act as deadlines, i.e. the obligation has to satisfied before the deadline occurs. This deadline can be generated by an agent action as a timekeeper.

Dedicated violation events can be introduced to indicate not only that an event has taken place without permission but also that an obligation was not fulfilled or that an undesirable event has taken place given the current state of affairs.

The domain fluents are used to describe the non-institutional state of the world. They keep track of what has happened in the system; i.e. a contract was signed, money was transferred, relevant details of the participants, etc..

2 Modelling Contract Law: A Case Study

Contract Cancellation under Japanese Law: We take our case study from Japanese contract law.

In our case study we look at contract cancellation. The question we aim to be able to answer is "When is it permissible to cancel a contract?". Although the specification is more general, the objective is to answer the question in the scenario presented in Figure 1 against the Articles presented in Figure 2.

InstAL Specification Contract execution and cancellation consists of various stages: (i) first a contract has to be established, (ii) then, if all parties adhere to it, the contract is executed (iii) a reminder can be sent if one of the parties misses a deadline (iv) if a party breaks the contract, the other party can cancel the contract, after which the transactions have to be undone. The Inst*AL* specification of these five phases, set-up, execution, reminding, cancellation and transaction reversal, can be found in Figures 3-8.

We discuss these stages in more detail later, but before doing so we need to declare the types, events and fluents needed in the specification. Their full declaration can be found in Figure 3. We name our institutional framework `legal` (Line 1). Our specification has only needs one type (Line 4) to represent the participants. The participant actions are represented as exogenous events (Lines 7-13). The contract institutions has five deadlines (dates) by which certain actions have to be performed. They are modelled

- **Article 412 (Time for Performance and Delay in Performance)**
  1. If any specified due date is assigned to the performance of an obligation, the obliger shall be responsible for the delay on and after the time of the arrival of such time limit.
  2. If any unspecified due date is assigned to the performance of a claim, the obliger shall be responsible for the delay on and after the time when he/she becomes aware of the arrival of such time limit.
  3. If no time limit is assigned to the performance of an obligation, the obliger shall be responsible for the delay on and after the time he/she receives the request for performance.
- **Article 540 (Exercise of Right to Cancel)**
  1. If one of the parties has a right to cancel in accordance with the provisions of the contract or law, the cancellation shall be effected by manifestation of intention to the other party.
  2. The manifestation of intention under the preceding paragraph may not be revoked.
- **Article 555 (Sale)**
  1. A sale shall become effective when one of the parties promises to transfer a certain real rights to the other party and the other party promises to pay the purchase money for it.
- **Article 541 (Right to Cancel for Delayed Performance)**
  1. In cases where one of the parties does not perform his/her obligations, if the other party demands performance of the obligations, specifying a reasonable period and no performance is tendered during that period, the other party may cancel the contract.

Fig. 2. Related Civil Law on contract cancellation (translated provided by Japanese Law Translation `http://www.japaneselawtranslation.go.jp`)

as exogenous events (Line 15 -19). Each institutional framework has a special event that will initialise it: for `legal`, we name this creation event `start` (Line 22). As described in section 1, for each exogenous events for which the model should account, we define a corresponding institutional event (Lines 25-32). Here, this means all the actions of the participants except for the reminder. Setting up the contract, `intSale`, results in two further institutional events, `transferReq` and `paymentReq`, whose purpose is to differentiate between buyers and sellers (Lines 26-27). The model also distinguishes four violation events, different from unpermitted events, to indicate that the buyer or the seller has not fulfilled their obligation with respect to the contract or its return policy (Lines 35-38). Apart from keeping track of power and permission for each of the events, the model also monitors the contract, the goods and the money involved in the transactions. This is done by the domain fluents defined in Lines 41-46.

The next step is to define the rules of the institutional framework, starting with setting up the contract (see Fig. 4). The triggering event of this phase is `sale`. A contract between buyer and seller (Line 53) is set up if `intSale` is empowered and permitted and if the seller has the goods and the buyer has the money (Line 53). The generation of `intSale` generates `transferReq` and `paymentReq` (Line 57) and initiates the `contract` fluent (Line 58). In turn, the two generated events set up the contractual obligations for the buyer and seller and give them the necessary power and permission to do so (Lines 61–70). The contractual obligations are automatically removed when the participants fulfil them or when the deadlines expire.

The next phase details the normal execution of a contract (see Fig. 5). When the seller (`giveGoods`) or the buyer (`makePayment`) satisfies their part of the contract (Lines 75–76), the corresponding institutional events change the owner of the money or goods and removes the power and permission of the execution events (Lines 78–84). If either participant fulfils their contractual obligations, irrespective of the deadline,

```
1   institution legal;
2
3   % Types
4   type Agent;
5
6   % exogeneous events
7   exogenous event sale(Agent,Agent); % Seller, Buyer
8   exogenous event giveGoods(Agent);
9   exogenous event makePayment(Agent);
10  exogenous event reminder(Agent);
11  exogenous event cancel(Agent);
12  exogenous event repayBuyer(Agent);
13  exogenous event returnSeller(Agent);
14
15  exogenous event handOverDeadline;
16  exogenous event paymentDeadline;
17  exogenous event handOverDeadline2;
18  exogenous event paymentDeadline2;
19  exogenous event returnDeadline;
20
21  % creation
22  create event start;
23
24  % institutional events
25  inst event intSale(Agent,Agent);   % Seller, Buyer
26  inst event transferReq(Agent);
27  inst event paymentReq(Agent);
28  inst event transfer(Agent);
29  inst event payment(Agent);
30  inst event cancellation(Agent);
31  inst event intrepayBuyer(Agent);
32  inst event intreturnSeller(Agent);
33
34  % violation
35  violation event contractViolationBuyer(Agent);
36  violation event contractViolationSeller(Agent);
37  violation event contractReturnViolationBuyer(Agent);
38  violation event contractReturnViolationSeller(Agent);
39
40  % fluents
41  fluent contract(Agent,Agent);  % Seller, Buyer
42  fluent cancelledContract(Agent,Agent);  % Seller, Buyer
43  fluent hasGood(Agent);
44  fluent hasMoney(Agent);
45  fluent paid;
46  fluent transferred;
```

Fig. 3. Declaration of types and events in the model

it should be impossible for the other party to cancel the contract (Lines 87–90). The occurrence of the deadlines `handOverDeadline` and `PaymentDeadline` permits the occurrence of reminders and terminates the permission for the deadline to occur again (Lines 92–96).

The reminder phase is specified in Fig. 6. From the previous phase we know that this phase can only occur without causing violation (from the reminder perspective) when the deadline for the contractual obligation has lapsed for one of the participants.

The cancellation phase specification is given in Fig. 7. Whenever one of the participants tries to cancel a contract (`cancel`) it is the status of the empowerment of the institutional event `cancellation` that determines whether the contract cancellation is valid or not (Line 116). The state of the contract may be changed from `contract` to `cancelled` (Lines 117–119). Once the contract is cancelled power and permission to cancel the contract is no longer required (Line 121). The power and permission to cancel

```
%-----------------------------------------------------------------
% Establish Contract
%-----------------------------------------------------------------
% the sale proceeds if sale is empowered and both participants have
% the correct assests
sale(Seller,Buyer) generates intSale(Seller,Buyer) if hasGood(Seller),
                             hasMoney(Buyer);

% sale split between buyer and seller responsibilities
intSale(Seller,Buyer) generates transferReq(Seller), paymentReq(Buyer);
intsale(Seller,Buyer) initiates contract(Seller,Buyer);

% obligations for buyer and seller specified
transferReq(Seller) initiates obl(transfer(Seller),handOverDeadline,
                                  contractViolationSeller(Seller));
paymentReq(Buyer) initiates obl(payment(Buyer),paymentDeadline,;
                                  contractViolationBuyer(Buyer));

% appropriate permission and power given
transferReq(Seller) initiates pow(transfer(Seller)),perm(transfer(Seller)),
                              perm(handOverDeadline);
paymentReq(Buyer) initiates pow(payment(Buyer)),perm(payment(Buyer)),
                            perm(paymentDeadline);
```

Fig. 4. Setting up a contract in Inst*AL*

```
%-----------------------------------------------------------------
% Executing Contract
%-----------------------------------------------------------------
giveGoods(Seller) generates transfer(Seller) if hasGood(Seller);
makePayment(Buyer) generates payment(Buyer) if hasMoney(Buyer);

transfer(Seller) initiates transferred, hasGood(Buyer)
                           if contract(Seller,Buyer);
transfer(Seller) terminates hasGood(Seller);
payment(Buyer) initiates paid, hasMoney(Seller) if contract(Seller,Buyer);
payment(Buyer) terminates hasMoney(Buyer);
transfer(Seller) terminates pow(transfer(Seller)),perm(transfer(Seller));
payment(Buyer) terminates pow(payment(Buyer)),perm(payment(Buyer));

% transfer or payment retracts permission to cancel for opposite party
transfer(Seller) terminates pow(cancellation(Byer)),perm(cancellation(Buyer)),
                            perm(cancel(Buyer)) if contract(Seller,Buyer);
payment(Buyer) terminates pow(cancellation(Seller)),perm(cancellation(Seller)),
                          perm(cancel(Seller)) if contract(Seller,Buyer);

handOverDeadline initiates perm(reminder(Buyer));
paymentDeadline initiates perm(reminder(Seller));

handOverDeadline terminates perm(handOverDeadline);
paymentDeadline terminates perm(paymentDeadline);
```

Fig. 5. Executing a contract in Inst*AL*

```
%-----------------------------------------------------------------
% Reminder
%-----------------------------------------------------------------
% give the agents a second change
reminder(Seller) initiates obl(transfer(Seller),handOverDeadline2,
                               contractViolationSeller(Seller)),
                           perm(handOverDeadline2) if not transferred;

reminder(Buyer) initiates obl(payment(Buyer),paymentDeadline2,
                              contractViolationBuyer(Buyer)),
                          perm(paymentDeadline2) if not paid;
```

Fig. 6. Reminders in the model

```
%-----------------------------------------------------------------
% Cancelling the contract
%-----------------------------------------------------------------
% Cancellation procudure: cancellation only empowered when violation
% occured and  only given to grieved agent

cancel(Agent) generates cancellation(Agent);
cancellation(Agent) terminates contract(Seller,Buyer);
cancellation(Agent) initiates cancelledContract(Seller,Buyer)
   if contract(Seller,Buyer);

cancellation(Agent1) terminates pow(cancellation(Agent)),
                                perm(cancellation(Agent)), perm(cancel(Agent));

% power and permission to cancel is given to appropriate agent
contractViolationSeller(Seller) initiates pow(cancellation(Agent)),
                                          perm(cancellation(Agent)),
                                          perm(cancel(Agent))
                                          if contract(Seller,Agent), paid;
contractViolationBuyer(Buyer) initiates pow(cancellation(Agent)),
                                        perm(cancellation(Agent)),
                                        perm(cancel(Agent))
                                        if contract(Agent,Buyer), transferred;
```

Fig. 7. Cancelling the contract in Inst*AL*

```
%------------------------------------------------------------------
% Return Policy
%-----------------------------------------------------------------
% if goods or money has changes owner they need to be returned when the
% contract is cancelled
cancellation(Agent) initiates perm(repayBuyer(Seller)),
                              pow(intrepayBuyer(Seller)),
                              perm(intrepayBuyer(Seller)),
                              obl(repayBuyer(Seller),returnDeadline,
                                  contractReturnViolationSeller(Seller))
                              if contract(Seller,Buyer), hasMoney(Seller);
cancellation(Agent) initiates perm(returnSeller(Buyer)),
                              pow(intreturnSeller(Buyer)),
                              perm(intreturnSeller(Buyer)),
                              obl(returnSeller(Buyer), returnDeadline,
                                  contractReturnViolationBuyer(Buyer))
                              if contract(Seller,Buyer), hasGood(Buyer);

repayBuyer(Seller) generates intrepayBuyer(Seller);
intrepayBuyer(Seller) initiates hasMoney(Buyer)
                        if cancelledContract(Seller,Buyer);
intrepayBuyer(Seller) terminates hasMoney(Seller)
                        if cancelledContract(Seller,Buyer);

returnSeller(Buyer) generates intreturnSeller(Buyer);
intreturnSeller(Buyer) initiates hasGood(Seller)
                         if cancelledContract(Seller,Buyer);
intreturnSeller(Buyer) terminates hasGood(Buyer)
                         if cancelledContract(Seller,Buyer);
```

Fig. 8. Modelling the return policy in Inst*AL*

```
%---------------------------------------------------------
% Initial state
%---------------------------------------------------------
initially pow(transferReq(seller)), pow(paymentReq(buyer)),
          perm(transferReq(seller)), perm(paymentReq(buyer)),
          perm(sale(seller,buyer)),perm(makePayment(buyer)),
          perm(giveGoods(seller)),hasGood(seller), hasMoney(buyer),
          pow(intSale(seller,buyer)), perm(intSale(seller,buyer));
```

Fig. 9. The initial state

```
observed(start,i00).
observed(sale(seller,buyer),i01).
observed(makePayment(buyer),i02).
observed(paymentDeadline,i03).
observed(handOverDeadline,i04).
observed(reminder(seller),i05).
observed(handOverDeadline2,i06).
observed(cancel(buyer),i07).
observed(repayBuyer(seller),i08).
```

Fig. 10. The complete trace of our contract example

the contract is given to a party in the contract when the other party fails to satisfy their part of the contract. In our model, failure is represented by a `contractViolation` event (Lines 125–132).

The final phase, the return policy, appears in Fig. 8. When a contract is cancelled, transferred goods and money must be returned to the original owner. Therefore, the occurrence of `cancellation` generates the permission and the obligation for the participants to return what they have received (Lines 139–150). When the buyer and seller return/repay the goods/money, ownership is adjusted accordingly (Lines 152–162).

With the five phases now specified, we just need the initial state to complete the model. In this case, this is the power and permission for the events that make up the first phase of the model (see Fig. 9).

Running the Model: Before we can run the model, we must declare the participating agents, by means of: `Agent: buyer seller`. The resulting ASP program can be used in a variety of ways. Without additional constraints, the model can compute all the possible ways exogenous events can take place and their consequences. From a more practical point of view, the model can be used to answer specific questions about the validity of an action/event given a sequence of events. While in most cases a complete set of events would be given, e.g. representing a specific case, the model is capable of reasoning about an incomplete sequence of events. Another use of the model is to verify the current state of affairs, e.g. is the contract still in place or has it been cancelled. Also, one can query the model for legal loopholes by asking if it is possible for a certain situation to occur.

While in litigation one often only takes the current status of rights and obligation between parties into account, there are situations, from example in tort or when reasoning about a prescription, where past and future statuses do matter. The model presented in paper, due the tracking the various states and partial sequences of events, allows for reasoning not only in the present but past and future as well.

ASP facts of the form `observed(event,time)` are added to represent complete or partial traces. A complete trace matching the scenario in Fig 1 displayed in Fig 10. It starts with the creation event followed by setting up the contract. After payment and the expiry of deadlines the seller (y) is reminded about his/her obligations. After that deadline expires the buyer cancels the contract. The trace finishes with seller repaying the buyer. The textual output from running the model is rather hard for humans to process. To make examining the model more user friendly Inst*Suite* provides trace diagrams that graphically represent the events and state changes over time. Figures 11 and 12 provide a visualization of the complete trace. States are denoted by circles and transition events are shown above the lines connecting states, with exogenous events in bold. For each state, all the fluents that are true in that state are listed and initiated fluents are in bold.

As we can see from the trace, the only violation that occurred was `contractViolationSeller(seller)`. No `viol(cancel(buyer))` occurred indicating that it was legal for buyer to cancel the contract.

3 Summary, Related and Future Work

In this paper we have demonstrated that is possible to use the concept institutional framework for the representation of and reasoning about legal state. We have shown that the same tools can be usefully employed to make machine processable equivalents of legislation available for the governance of and reference by software components.

The formal representation of legal frameworks by normative systems has been a subject of research for several decades: a comprehensive discussion appears in [10]. Contracts, specifically, have received much attention using different formalisms, although defeasible logic [4, 5] is considered particularly appropriate, along with various executable representations, such as RuleML in [4].

Herrestad in [7] argues that Sergot's formalisation of the library regulations is confronted with the Christholm paradox and that therefore logic programming formalism

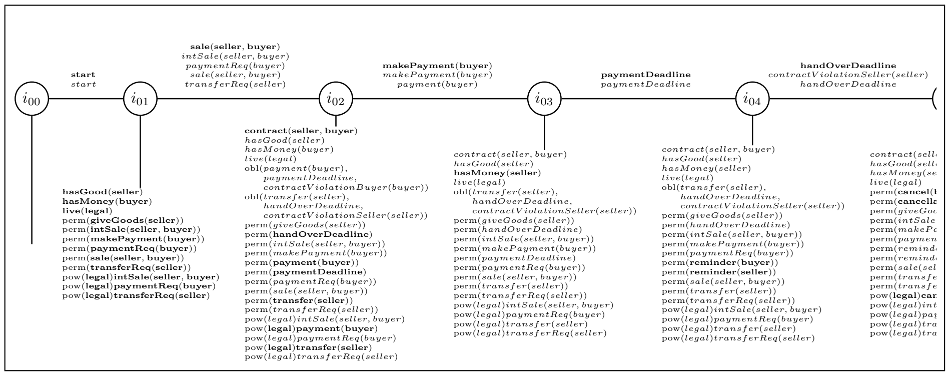

Fig. 11. The first 5 time instances of the example trace

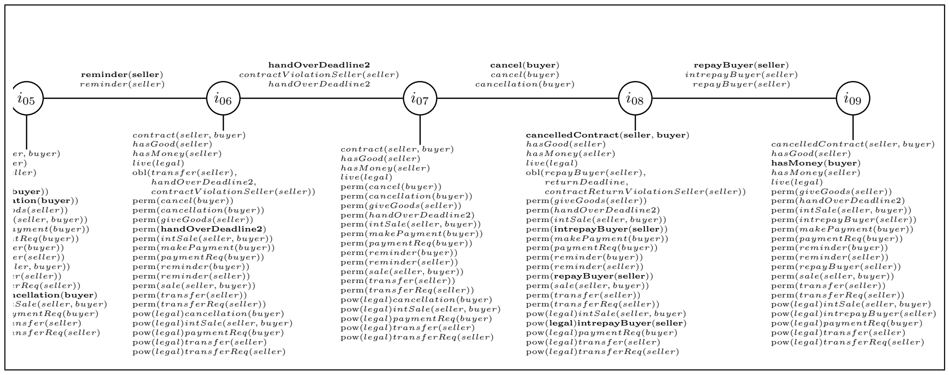

Fig. 12. The last 5 time instances of the example trace

are ill-suited for modelling this kind of knowledge. Herrestad claim is based the assumption that implication can be rewritten as a disjunction. In answer set programming, this is not the case as it uses negation as failure which if semantics differs from classical negation (see [2] for more information). Furthermore, with our approach there is no need to express that no disciplinary action is taken when books are returned on time. This is supported implicitly.

At this stage, we have completed an off-line specification that can be used to answer the question posed in section 2. While this is "just another model", we observe that conventionally in litigation, only the current status of rights and obligations between parties is usually considered. However, a potential benefit of the approach taken is the capacity to reason about not only the current status, but also the past (postdiction) and future status (prediction) [8]. Furthermore, the tools demonstrated (and under development—see below) may offer a way to bridge the practical gap between legal modelling and software agents.

Future work is proceeding on three fronts: (i) **contract recision:** this extends the case described here (ii) **on-line reasoning:** we are developing the means for agents to reason about institutional facts in a different domain, but using the same institutional model (iii) **model refinement:** as models become more complicated, they are harder to write and to debug: consequently, we are using an inductive logic programming application that given test cases and desired outcomes suggests new, and revises old, rules.

References

1. Owen Cliffe, Marina De Vos, and Julian Padget. Specifying and reasoning about multiple institutions. In *Coin*, volume 4386 of *LNAI*, pages 67–85. Springer Berlin / Heidelberg, 2007.
2. Marc Denecker. What's in a model? Epistemological analysis of logic programming. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004),*, pages 106–113, 2004.
3. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.
4. Guido Governatori. Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14(2-3):181–216, June-September 2005.
5. Guido Governatori and Antonino Rotolo. Defeasible logic: Agency, intention and obligation. In Alessio Lomuscio and Donald Nute, editors, *Deontic Logic in Computer Science*, volume 3065 of *LNAI*, pages 114–128, Berlin, 2004. Springer.
6. Davide Grossi, John-Jules Ch. Meyer, and Frank Dignum. The many faces of counts-as: A formal analysis of constitutive rules. *J. Applied Logic*, 6(2):192–217, 2008.
7. Henning Herrestad. Norms and formalization. In *ICAIL'91: Proceedings of the 3rd international conference on Artificial intelligence and law*, pages 175–184. ACM Press, 1991.
8. Luke Hopton, Owen Cliffe, Marina De Vos, and Julian Padget. Instql: A query language for virtual institutions using answer set programming. In *ClimaX*, pages 87–104, 2009.
9. John R. Searle. *The Construction of Social Reality*. Allen Lane, The Penguin Press, 1995.
10. Andrew J. I. Jones and Marek Sergot. On the characterization of law and computer systems: the normative systems perspective. In *Deontic logic in computer science: normative system specification*, pages 275–307. John Wiley and Sons Ltd., 1993.
11. Andrew J.I. Jones and Marek Sergot. A Formal Characterisation of Institutionalised Power. *ACM Computing Surveys*, 28(4es):121, 1996. Read 28/11/2004.
12. Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2010.