Embedding Landmarks and Scenes in a Computational Model of Institutions

Owen Cliffe, Marina De Vos, and Julian Padget

Department of Computer Science
University of Bath, BATH BA2 7AY, UK
{occ,mdv,jap}@cs.bath.ac.uk

Abstract. Over the last decade, institutions have demonstrated that they are a powerful mechanism to make agent interactions more effective, structured, coordinated and efficient. Different authors have tackled the problem of designing and verifying institutions from different angles. In this paper we propose a formalism that is capable of unifying and extending some of these approaches, as well as providing the necessary tools to assist in the design and verification processes. We demonstrate our approach with a non-trivial case-study.

1 Introduction

The concept of landmarks appears in [18] where they are used to identify a set of semantic properties relating to the state of a conversation, and which may furthermore be organized into sequences or patterns, while transition between landmarks is made by an appropriate sequence of one or more speech acts. A more detailed discussion follows in [12], where they are presented as propositions that are true in the state represented by the landmark (sic). The value of landmarks, and more specifically, their partial ordering into landmark patterns, is how they permit the identification of phases in a conversation protocol corresponding to the achievement of goals (and subgoals). Additionally, they form an integral part of realizing joint intention theory [7] as participants in a conversation interact with one another, *via* speech acts, to follow a common protocol and satisfy common goals. The utility of landmarks, from the electronic institution designer's perspective is their potential role in building a bridge [9, 1] between the rigidity of the protocols that feature in bottom-up design and the (relative) flexibility of norms that characterize top-down design.

The formal model put forward in [5] and its corresponding operationalization through Answer Set Programming (ASP) [3] aims to support the top-down design of electronic institutions through the provision of a domain-specific action language [17], called InstAL, tailored to the specification of institutions. Tools have been developed to translate InstAL into the SMODELS [14] syntax for processing by the answer set solver and furthermore the soundness and completeness of the institutional programs with respect to the formal model have been proven [4]. In this paper we explore the consequences of the correspondence between landmarks, as described in the literature, and the institutional states of our (executable) model, argue that the stronger logical framework of our formalism is advantageous and demonstrate the expressiveness of the Inst*AL* language through a non-trivial case-study.

2 The Institutional Framework

In this section we provide a brief description of our framework, starting with the formal model and following with the semantics. We then turn our attention to ASP as the underlying computational mechanism and the mapping from action language to ASP.

The Formal Model: Our model of an institution is a quintuple, $\mathcal{I} := \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$, comprising three disjoint sets:

- events $\mathcal{E}$, which can be either institutional (generated within the institution) or exogenous (caused by events outside of the scope of the institution) . In particular, we define a subset of the exogenous events as *creation events*, $\mathcal{E}_+$, which contain events which account for the creation of an institution and a subset of the institutional events as *dissolution events*, $\mathcal{E}_\times$.
- fluents $\mathcal{F}$, being the four distinguished sets of fluents — powers $\mathcal{W}$, permissions $\mathcal{P}$, obligations $\mathcal{O}$, domain-specific fluents $\mathcal{D}$ — that constitute the state of the institution and hence the basis for reasoning about the institution.
- and an initial state Δ comprising the initial set of fluents in the institution

and two relations $\mathcal{C}$ and $\mathcal{G}$ over $\mathcal{X} \times \mathcal{E}$, where $\mathcal{X} = 2^{(\mathcal{F} \cup \neg \mathcal{F})}$ and $\phi \in \mathcal{X}$ represents a set of conditions which must be met in a given state in order for either relation to have an effect.

- $\mathcal{C}$ defines under which circumstances fluents are initiated and terminated.
- $\mathcal{G}$ implements the *count-as* operation and defines under which conditions in the institutional state the occurence of a given event will result in the generation of one or more new events.

Semantics: The semantics of this framework are defined by traces of exogenous events. Each trace induces a sequence of institutional states, called a model. Starting from the initial state, the first exogenous event will, using the $\mathcal{G}$, generate a set of events. Each of these events will possibly affect the next state by means of the $\mathcal{C}$ relation. The combined effect results in the next state of the model. This process continues until all exogenous events in the trace have taken place.

ASP: In *answer set programming* ([3]) a logic program is used to describe the requirements that must be fulfilled by the solutions of a certain problem. The answer sets of the program, usually defined through (a variant/extension of) the stable model semantics [10], then correspond to the solutions of the problem. The programs consist of a set clauses with negation-as-failure in the body. Assumptions are verified by eliminating negation from the program using the Gelfond-Lifschitz reduction and to check if this new positive program sustains the assumptions made. Tools for obtaining answers sets are called answer set solvers. For our system we use the SMODELS [14] solver.

The Mapping: The mapping of each actual institution $\mathcal{I}$ into an answer set program consists of two parts: (i) P_{base} which is identical for each institution and handles the occurrence of observed events, the semantics of obligations and rules to maintain the commonsense inertia of fluents , and (ii) $P^*_{\mathcal{I}}$ which is specific to the institution being

modelled and represents the translation of it rules (norms and action semantics). Together they form the answer set program $P_{\mathcal{I}}$. In order to be able to use this program to reason about the institution, it is then combined with two other ASP programs: a trace program, containing a contraint on the length of traces of events being considered, and a query program expressing some constraint over the answer sets that shall be generated — the property or properties of the model that we wish to investigate.

InstAL : Our primary objective in this work is to be able to specify the behaviour of an institution in terms of its norms, and then to be able to test properties of the model of the institution thus defined. Consequently, we need a machine-processable representation. The engine for the verification is an answer set solver, so one approach would be to require the specification to be written in the input syntax for such a system, such as SMODELS, as outlined in [5]. However, while it may be useful for the designer to examine the code given to the answer set solver occasionally, it also necessarily contains low level support details that are less relevant to the task of institutional design. For this reason and because of the event-oriented nature of the specification, a domain-specific event language seems an appropriate medium, hence Inst*AL* .

We define the language Inst*AL* in order to simplify the process of specifying institutions. Individual institution specifications and multi-institution specifications are written as single Inst*AL* programs in a human-readable text format. These files can then be translated automatically into answer set programs that directly represent the semantics of the institutions specified in the original descriptions.

The language supports a simple set-based type system and syntax for the declaration of fluents, events, and institutions (bearing in mind the model also supports multi-institutional models as discussed in [6]). Normative fluents are pre-defined for power, permission and obligation. The designer may also specify static properties of an institution, that are initiated when the institution is created and never change. This provides a straightforward way to associate roles with institutions. Rather than give a formal syntax specification, for which there is not room here, we put forward and extended example in section 4 to illustrate the language features in a use-case. A detailed discussion of the Inst*AL* language can be found in [6, 4].

An Inst*AL* reasoning problem consists of the following:

1. One or more Inst*AL* institution descriptions each of which describes a single institution or a multi-institution.
2. A domain definition that grounds aspects of the descriptions. This provides the domains for types and any static properties referenced in the institution and multi-institution definitions.
3. A *trace program* which defines the set traces of exogenous events to investigate.
4. A *query program* which describes the desired property to validate with the Inst*AL* reasoning tool.

The reasoning process can be summarised as follows:

1. The Inst*AL* to ASP translator takes one or more single or multi-institution descriptions (in the Inst*AL* syntax described below), and domain definition files (described below) as input. Using these files, the translator generates a set of answer set programs which describe the semantics of the input institutions.

2. The translated institution programs along with a trace program and query program are then grounded by the LPARSE program (part of the SMODELS toolkit).
3. This grounded program description is then given as input to the SMODELS answer set solver. This produces zero or more answer sets. Each answer set corresponds to a possible model of the input institution for a given trace described by the trace program that matches the given query.
4. These answer sets may then be visualised and interpreted by the designer.

3 Landmarks and Scenes

As already discussed in the introduction, the essence of a landmark is a condition on a state in order for an action in some protocol to have effect. The relative sophistication of a landmark specification can be affected by the logic that is used to define the condition, but in many respects this is a technicality. For example [18] use first order logic augmented with modal operators for propositional attitudes and event sequences, [12] use dynamic propositional logic with modal operators from the previous work, while [9] (p.126) has atoms, implying the conjunction of positive values, within a Kripke model and [1] uses linear-time temporal logic. More important is the actual purpose of landmarks, as [12] states:

> Besides contributing to formal analyses of protocol families, the landmark-based representation facilitates techniques similar to partial order planning [13] for dynamically choosing the most appropriate action to use next in a conversation, allows compact handling of protocol exceptions, and in some cases, even allows short circuiting a protocol by opportunistically skipping some intermediate landmarks.

This highlights the relationship between agent actions and conventional AI planning and leads to the observation of the correspondence between landmarks and scenes (also mentioned in [9]). By scenes, we refer to the components of performative structure identified by Noriega [15] that are essentially sub-protocols of the larger institution or viewed bottom-up, an institution *may* be seen as the composition of numerous protocols that help agents achieve various sub-goals. What it is important to observe about Noriega's (and later in [16]) definition of the performative structure is how various conditions are imposed on the transitions from one scene to another, typically constraining the number and role of the agents that may move. A scene essentially encapsulates a self-contained protocol whose purpose is to achieve some sub-goal of the institution contributing to the objective of using the institution in the first place.

From this perspective, we can now turn to the relationship between our formalism and both landmarks and scenes, having established that both concepts serve to identify some (final) state in which a condition (capturing some institutional sub-goal) has been satisfied. Returning to the relations that drive our formalism (see section 2), the event generation function serves to create institutional facts, while the consequence relation focuses attention on the initiation and termination of fluents. The function is expressed as $\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$. Where the first set in the range of the function describes which fluents are initiated by the given event and the second set represents those fluents terminated by the event. We use the notation $\mathcal{C}^{\uparrow}(\phi, e)$ to denote the fluents that are

initiated by the event e in a state matching ϕ and the notation $\mathcal{C}^{\downarrow}(\phi, e)$ to denote those terminated by event e in a state matching ϕ.

From the description of event generation and the consequence relation, it can be seen that fluents are initiated and terminated in respect of an event and some conditions on the state of the institution. This corresponds exactly with the notion of landmark, in that an event takes the institution into a new state *but* this is predicated on the current state — that is, a condition. Thus landmarks arise naturally from our formalization and furthermore, the condition language would appear to be richer than in some earlier work because the condition may contain both positive and negative information, including the use of negation as failure and hence non-monotonic reasoning, since these are basic properties of answer set semantics. Our conclusion therefore is that our formalism provides landmarks for free and, thanks to ASP semantics, enriches the landmark description language over earlier examples.

In the literature cited above, landmarks appear to be restricted to speech acts, that is messages from participating agents. Our model goes further, as we also consider exogenous events that do not originate from participating agents or from institutional events. This makes our approach a convenient tool for reasoning with scenes, where the transition between the various scenes does not necessarily depend on agents' actions. Instead the transition markers could be linked to exogenous events which are taken into account when the institution reaches a certain state. At this point the consequence relation could be used to set the powers and permissions (and so the behaviour) of the participating agents. The Dutch auction protocol detailed in the next section uses this technique to distinguish between the various phases/scenes of the protocol.

4 The Dutch Auction Protocol

Informal Description of Dutch Auction: In this protocol a single agent is assigned to the role of auctioneer, and one or more agents play the role of bidders. The purpose of the protocol as a whole is either to determine a winning bidder and a valuation for a particular item on sale, or to establish that no bidders wish to purchase the item. The protocol is summarised as follows:

1. Round starts: The auctioneer selects a starting price for the item and informs each of the bidders present of this price. The auctioneer then waits for a given period of time for bidders to respond.
2. Upon receipt of the starting price, each bidder has the choice as to whether to send a message indicating their desire to bid on the item at that price, or to send no message indicating that they do not wish to bid on the item.
3. At the end of the prescribed period of time, if the auctioneer has received a single bid from a given agent, then the auctioneer is obliged to inform each of the participating agents that this agent has won the auction.
4. If no bids are received at the end of the prescribed period of time, the auctioneer must inform each of the participants that the item has not been sold.
5. If more than one bid was received then the auctioneer must inform each agent that a conflict has occurred.
6. In the case where the item is sold the protocol is finished.

7. In the case that no bids are received then the auctioneer may either start a new round of bidding at a lower price, or withdraw the item from sale.
8. In the case where a conflict occurs then the auctioneer must re-open the bidding at a higher price and start the round again in order to resolve the conflict.

We focus on the protocol for the round itself (items 1-6). In our description below we omit from the messages a definition of the item in question and the starting price. While the inclusion of these aspects in the protocol is possible, their inclusion does not change the structure of the protocol round so we leave them out for simplicity.

In the following paragraphs we go through the Inst*AL* code step by step. The full listing can be found in Figures 1 and 2. Each line of Inst*AL* code is labelled with DAR-FigureNr-LineNr for ease of reference.

The first lines indicate the name of the institution (DAR-1-1) and the types of agents, Bidder (DAR-1-2) and Auctioneer (DAR-1-3) that may participate in the institution. These types are used as placeholders in the Inst*AL* rules for the agents participating in a particular instance of the institution, then when instantiated all rules are grounded appropriately. The institution is created by one creation event `createdar` as specified by rule DAR-1-4.

Based on the protocol description above, the following agent messages are defined (DAR-1-8 – DAR-1-12): the auctioneer announces a price to a given bidder (`annprice`), the bidder bids on the current item (`annbid`), the auctioneer announces a conflict to a given bidder (`annconf`) and the auctioneer announces that the item is sold (`annsold`) or not sold (`annunsold`) respectively. Each exogenous action has a corresponding institutional event (DAR-1-16 – DAR-1-20 which accounts for a valid execution of the physical action performed. In all cases the two events are linked by an unconditional generates statement in the description (DAR-2-29,DAR-2-32,DAR-2-37,DAR-2-38,DAR-2-39).

In addition to the agent actions we also include a number of time-outs indicating the three external events (which are independent of agents' actions) that affect the protocol. For each time-out we define a corresponding institutional event suffixed by `dl` indicating a deadline in the protocol:

`priceto`, `pricedl`: A time-out indicating the deadline by which the auctioneer must have announced the initial price of the item on sale to all bidders. (DAR-1-5 and DAR-1-13)

`bidto`, `biddl`: A time-out indicating the expiration of the waiting period for the auctioneer to receive bids for the item (DAR-1-6 and DAR-1-14).

`desto`, `desdl`: A time-out indicating the deadline by which the auctioneer must have announced the decision about the auction to all bidders (DAR-1-7 and DAR-1-15).

We assume that the time-outs will occur in the order specified (that is, due to their durations it is impossible for this to be otherwise). We use the corresponding institution events in the protocol description and constrain the order in which they are empowered in the institution to ensure that while the exogenous events may occur in any order, the institution event may only occur once in each iteration and in the order specified (DAR-2-52 to DAR-2-59).

We define a single additional institution event `alerted(Bidder)` (DAR-1-23) that represents the event of a bidder being validly notified of the result of the auction. We

```
institution dutch;                                          (DAR-1-1)

type Bidder;                                                (DAR-1-2)
type Auct;                                                  (DAR-1-3)

create event createdar;                                     (DAR-1-4)

exogenous event priceto;                                    (DAR-1-5)
exogenous event bidto;                                      (DAR-1-6)
exogenous event desto;                                      (DAR-1-7)

exogenous event annprice(Auct,Bidder);                      (DAR-1-8)
exogenous event annbid(Bidder,Auct);                        (DAR-1-9)
exogenous event annconf(Auct,Bidder);                      (DAR-1-10)
exogenous event annsold(Auct,Bidder);                      (DAR-1-11)
exogenous event annunsold(Auct,Bidder);                    (DAR-1-12)

inst event pricedl;                                        (DAR-1-13)
inst event biddl;                                          (DAR-1-14)
inst event desdl;                                          (DAR-1-15)

inst event price(Auct,Bidder);                             (DAR-1-16)
inst event bid(Bidder,Auct);                               (DAR-1-17)
inst event conf(Auct,Bidder);                              (DAR-1-18)
inst event sold(Auct,Bidder);                              (DAR-1-19)
inst event unsold(Auct,Bidder);                            (DAR-1-20)

dest event badgov;                                         (DAR-1-21)
dest event finished;                                       (DAR-1-22)

inst event alerted(Bidder);                                (DAR-1-23)

fluent onlybidder(Bidder);                                 (DAR-1-24)
fluent havebid;                                            (DAR-1-25)
fluent conflict;                                           (DAR-1-26)

initially pow(price(A,B)), perm(price(A,B)),
          perm(annprice(A,B)),
          perm(badgov),pow(badgov),
          perm(pricedl),pow(pricedl),
          perm(priceto),
          perm(biddl),
          perm(bidto),
          perm(desto);                                     (DAR-1-27)
```

Fig. 1. InstAL for the Dutch Auction Round Institution Part 1

additionally specify a dissolution event `finished` (DAR-1-22) that indicates the end of the protocol.

For the sake of simplicity, we do not focus in detail on the effects of the auctioneer violating the protocol. Instead we define a dissolution institutional event `badgov` (DAR-1-21) that accounts for a*any* instances in which the auctioneer has violated the protocol. Once an auctioneer has violated the protocol, we choose to treat the remainder of the protocol as invalid and dissolve the institution.

```
initially obl(price(A,B),pricedl,badgov);                                    (DAR-2-28)
annprice(A,B) generates price(A,B);                                          (DAR-2-29)
price(A,B) terminates pow(price(A,B));                                       (DAR-2-30)
price(A,B) initiates pow(bid(B,A)),perm(bid(B,A)),perm(annbid(B,A));         (DAR-2-31)

annbid(A,B) generates bid(A,B);                                              (DAR-2-32)
bid(B,A) terminates pow(bid(B,A)),perm(bid(B,A)),perm(annbid(B,A));          (DAR-2-33)
bid(B,A) initiates havebid,onlybidder(B) if not havebid;                     (DAR-2-34)
bid(B,A) terminates onlybidder(_) if havebid;                                (DAR-2-35)
bid(B,A) initiates conflict if havebid;                                      (DAR-2-36)

annsold(A,B) generates sold(A,B);                                            (DAR-2-37)
annunsold(A,B) generates unsold(A,B);                                        (DAR-2-38)
annconf(A,B) generates conf(A,B);                                            (DAR-2-39)
biddl terminates pow(bid(B,A));                                              (DAR-2-40)
biddl initiates pow(sold(A,B)),pow(unsold(A,B)),
       pow(conf(A,B)), pow(alerted(B)),perm(alerted(B));                     (DAR-2-41)
biddl initiates perm(annunsold(A,B)),perm(unsold(A,B)),
       obl(unsold(A,B),desdl,badgov) if not havebid;                         (DAR-2-42)
biddl initiates perm(annsold(A,B)),perm(sold(A,B)),
        obl(sold(A,B), desdl, badgov) if havebid, not conflict;              (DAR-2-43)
biddl initiates perm(annconf(A,B)),perm(conf(A,B)),
        obl(conf(A,B), desdl, badgov) if havebid, conflict;                  (DAR-2-44)
unsold(A,B) generates alerted(B);                                            (DAR-2-45)
sold(A,B) generates alerted(B);                                              (DAR-2-46)
conf(A,B) generates alerted(B);                                              (DAR-2-47)
alerted(B) terminates pow(unsold(A,B)), perm(unsold(A,B)),
 pow(sold(A,B)), pow(conf(A,B)), pow(alerted(B)),
 perm(sold(A,B)), perm(conf(A,B)), perm(alerted(B)),
 perm(annconf(A,B)),perm(annsold(A,B)),perm(annunsold(A,B));                 (DAR-2-48)
desdl generates finished if not conflict;                                    (DAR-2-49)
desdl terminates havebid,conflict,perm(annconf(A,B));                        (DAR-2-50)
desdl initiates pow(price(A,B)), perm(price(A,B)),
 perm(annprice(A,B)), perm(pricedl),pow(pricedl),
 obl(price(A,B),pricedl,badgov) if conflict;                                 (DAR-2-51)
priceto generates pricedl;                                                   (DAR-2-52)
pricedl terminates pow(pricedl);                                             (DAR-2-53)
pricedl initiates pow(biddl);                                                (DAR-2-54)

bidto generates biddl;                                                       (DAR-2-55)
biddl terminates pow(biddl);                                                 (DAR-2-56)
biddl initiates pow(desdl);                                                  (DAR-2-57)

desto generates desdl;                                                       (DAR-2-58)
desdl terminates pow(desdl);                                                 (DAR-2-59)
```

Fig. 2. InstAL for the Dutch Auction Round Institution Part 2

Once the institution has been created, the auctioneer will receive power and permission to announce prices. We also provide empowerment and permission for the dissolution event `badgov`. Furthermore all deadlines are permitted but only pricing is empowered. This is specified by DAR-1-27.

The rules of the institution are driven by the occurrence of the time-outs described above and hence may be broken down in to three phases as follows:

1. In the first phase of the protocol the auctioneer must issue price statements to each of the bidders. We represent this in the protocol by defining an initial obligation on the auctioneer to issue a price to each bidder before the price deadline (DAR-2-28). Once this has taken place, the auctioneer is no longer permitted to issue a price (DAR-2-30).
   Once a price has been sent to the bidder, the bidder is empowered and permitted to bid in the round (note that we permit both the action of validly bidding itself, `bid(B,A)`, as well as the action of sending the message which may count as bidding, `annbid(B,A)` (DAR-2-31).
2. In the second phase of the protocol, bidders may choose to submit bids. These must be sent before the bid time-out event. In order to account for the final phase of the protocol, we must capture the cases when one bid, no bids or multiple bids (a conflict) occur. In addition, in a given round, we must also take into account that bids may be received asynchronously from different agents over a period of time. In order to capture which outcome of the protocol has occurred we use three domain fluents (DAR-1-24 – DAR-1-26) to record the state of the bidding: `onlybidder(Bidder), havebid, conflict`.
   The first of these fluents denotes the case where a single bid has been received and no others (and records the bidder which made this bid), the second fluent records cases where one or more bids have been received and the third records cases where more than one bid has been received.
   These fluents are determined in the second phase of the protocol using DAR-2-34, DAR-2-35 and DAR-2-36. The first rule accounts for the first bid that is received, and is only triggered if no previous bids have been made. The second rule accounts for any further bids and terminates the `onlybidder` fluent when a second bid is received. The final rule records a conflict if a bid is received and a previous bid has occurred.
   Once a bid has been submitted we do not wish to permit an agent to submit further bids, or for those further bids to be valid. In order to account for this we have line DAR-2-33.
3. In the third and final phase of the protocol the auctioneer must notify the bidding agents of the outcome of the auction. This phase is brought about by the occurrence of the `biddl` event which denotes the close of bidding. In order to account for this, we terminate each agents' capacity to bid further in the auction (DAR-2-40) and correspondingly initiate the auctioneer's power to bring about a resolution to the auction (DAR-2-41). To do so, we create an obligation upon the auctioneer to issue the right kind of response (`sold, unsold, conflict`) depending on outcome of the previous phase (`havebid,conflict`) before the next deadline (`desdl`) is announced. This is encoded by DAR-2-42 – refdar:obl2. For each outcome, the auctioneer is obliged and permitted to issue the appropriate response to every bidding agent before the decision deadline. If an auctioneer fails to issue the correct outcome to any agent before the final deadline then a violation will occur. The protocol follows these notifications using DAR-2-45 – DAR-2-46.
   Once an agent has been notified we wish to prohibit the auctioneer from notifying that agent again. We do this by introducing a rule which terminates the auctioneer's

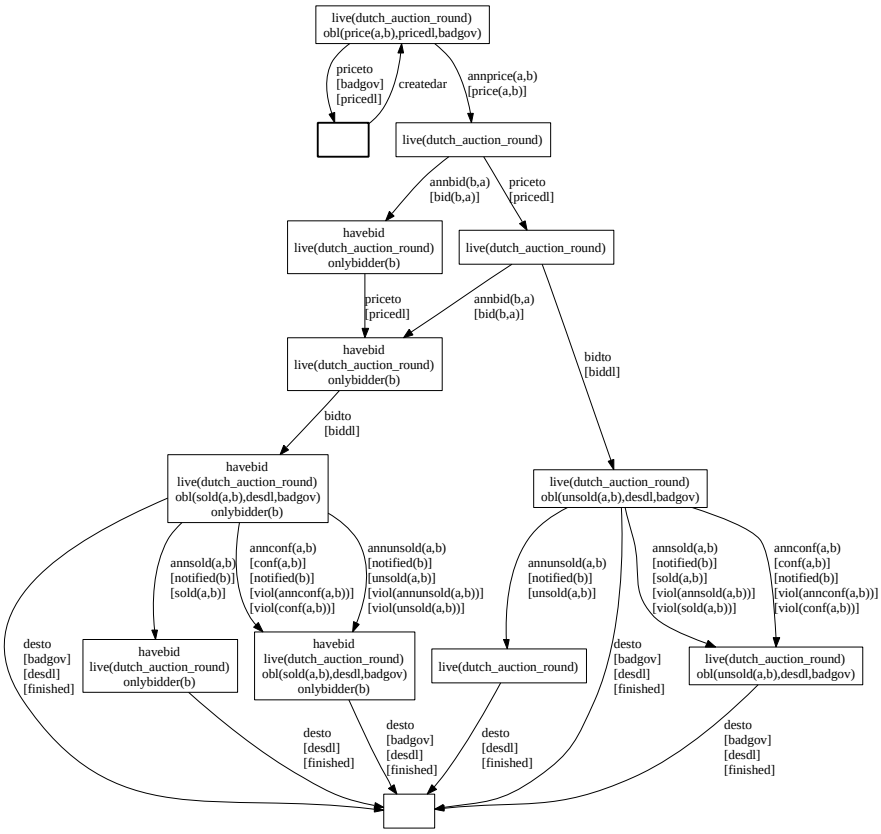

Fig. 3. States of the auction round for a single bidder

power and permission to issue more than one notification to any one agent (DAR-2-48).

Finally, when the deadline expires (the exogenous event `desto` triggers `desdl`) and either the protocol ends or the bidders have created a conflict. In the former case, DAR-2-49 ensures dissolution of the institution. In the conflict case, the auctioneer must re-open the bidding using a new round. We represent this by adding two lines. The first terminates the intermediate fluents which were used to represent the outcome of the protocol (`havebid` and `conflict`). This is established by DAR-2-50

. The second (DAR-2-51), initiates the obligation for the auctioneer to re-open the round by issuing a price to the bidders and all associated powers and permissions.

Verification: Once we have the Inst*AL* description of our institution, we can obtain an ASP program as described in Section 2. This program may then be combined with a trace program and query, allowing us to query properties and determine possible outcomes of this protocol.

The simplest type of verification procedure is to execute the program with no query. In this case all possible traces of the protocol will be provided as answer sets of the translated program.

Each answer set represents all possible sequences of states which may occur in the model and these may in turn be used to visualise all reachable states of the protocol (for a given number of agents). In order to execute the protocol we need to ground it with an auctioneer a and a bidder b. We could execute the translated program as is, however the answer sets of the program would include *all* traces of the protocol, including those containing actions which have no effect. Transitions of this kind may be of interest in some cases (we may be interested in the occurrence of associated violations for instance) however in this case we choose to omit them in order to reduce the number of answer sets to analyse. This can be achieved by specifying a query program which limits answer sets only to those containing traces in which a change of state occurs. For the technical details on this query program, see [4].

Solving the translated program with the associated query program yields a total of 60 answer sets corresponding to each possible trace where an effect occurs in each transition. By extracting the states from the answer set we may generate a graphical representation of the transition system which the protocol creates.

In order to include all possible states of the protocol we must select a large enough upper bound for the length of traces such that all possible states are reached. In general the selection of this upper bound depends on the program and query in question and it should be noted that the answer sets of the program represent *only* those solutions to the query which can be found in the given trace length.

In the case of the auction protocol examined here we had to establish this upper bound by the somewhat unsatisfactory process of iterating the solver process and determining the number states until no more states were found. For the example above, with only two agents, the longest traces which yield new states are of length 7, resulting in 33 answer sets.

Figure 3 illustrates all possible states for a single round of the protocol with one bidder (for a larger number of bidders, the state space will be considerably larger, growing exponentially with their number). Note that as there is only one bidder participating in the protocol conflicts cannot occur. For the sake of clarity we omit fluents relating to powers and permissions from the figure.

Further verification: In the above protocol we stated that when there was a conflict in the bidding for the protocol (that is, when two or more bidders issue valid bids) that the bidding should re-open. In order to ensure that this new round continues as before we must ensure that the institutional state at the beginning of a re-opened round is the same as the institutional state when the original round opened.

This property may be specified as a query program in our framework, as we now describe. In this case we are only interested in traces where a conflict has occurred. We specify this by adding the following constraints to the query program:

$$\texttt{hadconflict} \leftarrow \texttt{holdsat}(\texttt{conflict}, \texttt{I}), \texttt{instant}(\texttt{I}).$$
$$\bot \leftarrow \textbf{not}\ \texttt{hadconflict}.$$

The first rule states that if there is any state where the conflict fluent occurs, then the literal hadconflict should be included in the answer set. The second rule states that we should not include any answer sets where the literal hadconflict is not included.

We are also only interested in traces where the protocol is re-started and bidding is re-opened. We add this constraint in a similar way, using two rules as follows:

$$\begin{aligned} \texttt{restarted} \leftarrow\ & \texttt{occurred(desdl, I)}, \\ & \texttt{holdsat(conflict, I)}, \texttt{instant(I)}. \\ \bot \leftarrow\ & \textbf{not}\ \texttt{restarted}. \end{aligned}$$

The first of these rules state that if the desdl event has occurred at any time we include the literal restarted in our answer set and the second rule states that we should only include answer sets where this literal is included.

In order to determine the fluents (if any) which differ between a state following the creation of the institution and a state following a protocol re-start, we mark these fluents using the literals startstate(F) indicating that fluent F is true in the start state of this trace, and restartstate(F) indicating that the fluent F was true in a state following a protocol re-start.

Literals of the form startstate(F) are defined using the following rule:

$$\begin{aligned} \texttt{startstate(F)} \leftarrow\ & \texttt{holdsat(F, I1)}, \\ & \texttt{occurred(createdar, I0)}, \\ & \texttt{next(I0, I1)}, \texttt{ifluent(F)}. \end{aligned}$$

Which states that F is a fluent in the start state, if F holds at time instant I1 and creation event createdar occurred at instant I0 and that instant I1 immediately follows instant I0.

We similarly define the fluents that hold in the re-start state with the rule:

$$\begin{aligned} \texttt{restartstate(F)} \leftarrow\ & \texttt{holdsat(F, I1)}, \texttt{occurred(desdl, I0)}, \\ & \texttt{holdsat(conflict, I0)}, \texttt{next(I0, I1)}, \texttt{ifluent(F)}. \end{aligned}$$

which states that F holds in the restart state, if it held in the state I1 which immediately followed the occurrence of the decision deadline desdl when a conflict held in that state.

We then define the following rules which indicate the differences between the start state and the re-start state:

$$\begin{aligned} \texttt{missing(F)} &\leftarrow \texttt{startstate(F)}, \texttt{notrestartstate(F)}, \texttt{ifluent(F)}. \\ \texttt{added(F)} &\leftarrow \texttt{restartstate(F)}, \texttt{notstartstate(F)}, \texttt{ifluent(F)}. \end{aligned}$$

These rules indicate that a fluent is present in the start state, but missing from the restart state (indicated by missing(F)), or missing in the start state, but present in the restart state (indicated by added(F)) respectively.

Finally we define the query constraint, in this case we are only interested in traces where a difference occurs between the start state and the restart state. We add these constraints using following rules:

$$\begin{aligned} \texttt{invalid} &\leftarrow \texttt{missing(F)}, \texttt{ifluent(F)}. \\ \texttt{invalid} &\leftarrow \texttt{added(F)}, \texttt{ifluent(F)}. \\ \bot &\leftarrow \textbf{not}\ \texttt{invalid}. \end{aligned}$$

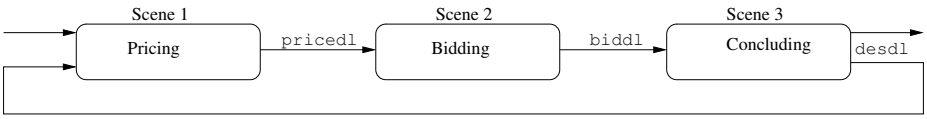

Fig. 4. Landmarks in the Dutch Auction round

The first two rules state that if a fluent `F` is either `missing` or `added`, then the literal `invalid` is true. The third rule constrains answer sets of the program to only those containing the literal `invalid`.

These rules, when combined with the translated program of the institution allow us to determine which fluents have changed between the start state and end state of the protocol.

Given the translated program and the query program described above, we obtain no answer sets for the protocol as defined, indicating that it is indeed the case that there are no fluents which differ in the state following a protocol restart and the state following the creation of the institution. This result is consistent with the original description of the protocol and will permit subsequent rounds following a conflict to continue in the same way as the original round. The same query holds true for auctions including three or four bidders.

The Scene Perspective: Although the Inst*AL* language does not explicitly allow for the definition of scenes (i.e. no special constructs are available), it is straightforward to achieve this with the available language constructs. The auction protocol discussed above, can be seen as composed of three scenes each marked by the occurrence of a deadline (except for the start of the protocol). Figure 4 provides the scene transition diagram. Each of these deadlines is the result of an exogenous event generated by the environment (e.g. DAR-2-52). The occurrence of such a deadline, changes the empowerment and permissions of the agents involved in the protocol (e.g. DAR-2-50). Rules are provided to assure the correct transition through the scenes (e.g. DAR-2-53).

5 Conclusions

Due to page restrictions, we are unable to include a separate overview of related work. Instead we have added pointers to related work wherever possible throughout the presentation. An extensive discussion of the relation between our framework and other normative systems, such as [2, 8, 11, 17, 19] to name but a few, can be found in [5, 4].

In this article we have demonstrated that the formal system described in [5] can easily deal with non-trivial institutions. Furthermore, we have shown that our characterisation can deal directly with landmarks and scenes, thus linking it more clearly with earlier work on institutional specification.

References

[1] H. Aldewereld. *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*. PhD thesis, Utrecht, 2007.

[2] Alexander Artikis. *Executable Specification of Open Norm-Governed Computational Systems*. PhD thesis, Department of Electrical & Electronic Engineering, Imperial College London, Sept. 2003.

[3] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[4] O. Cliffe. *Specifying and Analysing Institutions in Multi-Agent Systems Using Answer Set Programming*. PhD thesis, Dept. Computer Science, University of Bath, June 2007.

[5] O. Cliffe, M. De Vos, and J. A. Padget. Answer set programming for representing and reasoning about virtual institutions. In K. Inoue, K. Satoh, and F. Toni, editors, *CLIMA VII*, volume 4371 of *Lecture Notes in Computer Science*, pages 60–79. Springer, 2006.

[6] O. Cliffe, M. De Vos, and J. A. Padget. Specifying and reasoning about multiple institutions. In J. Vazquez-Salceda and P. Noriega, editors, *COIN 2006*, volume 4386 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2007.

[7] P. R. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[8] M. Colombetti, N. Fornara, and M. Verdicchio. The role of institutions in multiagent systems. In *Proceedings of the Workshop on Knowledge based and reasoning agents, VIII Convegno AI*IA 2002, Siena, Italy*, 2002.

[9] V. Dignum. *A Model for Organizational Interaction*. PhD thesis, Utrecht, 2004.

[10] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of fifth logic programming symposium*, pages 1070–1080. MIT PRESS, 1988.

[11] L. Kamara, A. Artikis, B. Neville, and J. Pitt. Simulating computational societies. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Proceedings of workshop on engineering societies in the agents world (esaw)*, LNCS 2577, pages 53–67. Springer, 2003.

[12] S. Kumar, M. J. Huber, P. R. Cohen, and D. R. McGee. Toward a formalism for conversation protocols using joint intention theory. *Computational Intelligence*, 18(2):174–228, 2002. doi:10.1111/1467-8640.00187.

[13] S. Minton, J. Bresina, and M. Drummond. Total order and partial order planning: A comparative analysis. *Journal of Artificial Intelligence Research*, 2:227–262, 1994.

[14] I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International Conference on Logic Programing and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 420–429, Berlin, July 28–31 1997. Springer.

[15] Pablo Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona, 1997.

[16] J. A. Rodríguez-Aguilar. *On the Design and Construction of Agent-mediated Institutions*. PhD thesis, Universitat Autonoma de Barcelona, 2001.

[17] M. Sergot. (C+)++: An Action Language For Representing Norms and Institutions. Technical report, Imperial College, London, Aug. 2004.

[18] I. Smith, P. Cohen, J. Bradshaw, M. Greaves, and H. Holmback. Designing conversation policies using joint intention theory. In *Proceedings of International Conference on Multi Agent Systems*, pages 269–276, 1998. doi:10.1109/ICMAS.1998.699064.

[19] P. Yolum and M. P. Singh. Flexible protocol specification and execution: applying event calculus planning using commitments. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 527–534. ACM Press, 2002.