

Specification and Verification of Institutions through Status Functions ^{*}

Francesco Viganò¹ and Marco Colombetti^{1,2}

¹ Università della Svizzera italiana, via G. Buffi 13, 6900 Lugano, Switzerland
{francesco.vigano, marco.colombetti}@lu.unisi.ch,

² Politecnico di Milano, piazza Leonardo Da Vinci 32, Milano, Italy
marco.colombetti@polimi.it

Abstract. Institutions have been proposed as a means to regulate open interaction systems by introducing a set of norms (involving deontic positions like authorizations, obligation, prohibition and permissions) and to define the ontology of the context in which agents interact. To better clarify the interdependence existing among deontic positions and the ontology defined by each institution, in this paper we propose to model institutions in terms of status functions imposed on agents and defined as aggregates of deontic positions. We present a metamodel which describes the concepts necessary to specify an institution and FIEVeL, a language that can be used to formalize institutions. Finally, we discuss how to automatically translate FIEVeL specifications into the input language of the SPIN model checker and the kind of properties that it is possible to check.

1 Introduction

Following the ideas presented in [17], institutions have been put forward as a means for regulating open interaction systems where agents' internal states cannot be accessed or agents are implemented by different parties. *Norms* play a fundamental role in the design and management of open systems, because they create positive expectations in the outcomes of such systems and make more predictable the behavior of other agents which are assumed to be autonomous. For this reason, in [7, 3, 13, 10] the authors propose to model interaction systems in terms of *electronic institutions*, a formalism developed to design protocols and which can be used to automatically enforce them as explained in [6]. From a different perspective, but with similar objectives, also in [20, 22] *normative systems* have been proposed to describe agents behavior in terms of obligations, prohibitions, and permissions when it is not possible to assume that agents will behave according to the expected behavior.

Following the analysis of institutional reality presented in [21], in [9, 24] we have suggested that institutions not only define norms, but also the *ontology* of the interaction context. For instance, the institution of the English Auction defines the very concept of winning an auction, which also implies that the winner ought to follow a set of norms. In [9, 24] institutional attributes assigned to entities and norms are combined to describe

^{*} Supported by Swiss National Science Foundation project 200020-109525, "Artificial Institutions: specification and verification of open distributed interaction frameworks."

the interaction context and the protocol as in [4, 20]. Institutions have been essentially applied to define protocols, therefore we will exemplify our approach with examples taken by a widely used interaction protocol, the English Auction, which allows us to exemplify and compare our approach with other attempts to formalize institutions.

While in [4, 9, 20, 24] institutional states are described and are assumed to exist independently of agents' deontic positions (authorizations, obligations, etc.), following Searle [21] we think that, with the exception of institutional facts related with the creation of meaning, institutional facts cannot be described or exist without deontic positions. In fact, when an agent modifies or creates a new institutional fact, it is actually modifying an authorization or creating new obligations for another agent or for itself. For instance, when the auctioneer declares a new current price, participants are authorized only to make higher bids.

Following Searle [21], institutional facts are built thanks to the ability of agents to collectively impose new statuses, named *status functions*¹, on objects, events or agents. When a new status function is imposed on an agent (or an object) by a community of agents, it holds certain positions and performs certain functions independently of its physical features. Moreover, since institutional objects represent the continuous possibility of agents' activities [21], imposed status functions become manifest only when they are necessary to perform actions. Therefore, when agents impose a new status function, they are actually creating new possibilities of action for agents themselves.

Starting from this point, we propose to model all institutional facts, even those imposed on events and objects, in terms of status functions imposed on agents, which are named *agent status functions*. In particular, we describe agent status functions in terms of deontic positions, which represent what actions are authorized, obliged, forbidden, or permitted for an agent. In doing so, institutional events can be characterized in terms of what status functions are imposed or revoked, which helps to clarify how each institutional event changes agents' deontic positions.

Once we have formalized an institution, we have also to ensure that our specification is sound and allows agents to reach the desired states of affairs. Furthermore, as soon as institutions become complex, without the aid of an automated techniques it is prohibitive to foresee all possible evolutions and states in which the institution may evolve. For this reason in this paper we also consider how to verify institutions by applying model-checking techniques [2, 19]. Automated formal verification should be considered as an important step in the development of institutions, because it can increase the reliability of institutions by ensuring that agents will be provided with all the needed powers to fulfill their objectives and that they will not be subject to contradictory or undesired norms.

The remainder of this paper is so structured: in Section 2 we define a metamodel of institutional reality based on the concept of status functions. In Section 3 we briefly present the syntax and the semantics of FIEVeL, a language that can be used not only to model artificial institutions, but also also to verify such specifications. In Section

¹ As in [21] we use the term *function* to refer to a position or an action for which an agent or a thing is specially fitted or to a position imposed on agents or things independently of their specific physical features (status function). Therefore, we do not use the term function in its mathematical sense.

4 we show how FIEVeL specifications can be translated into Promela (*Process Meta Language*), the input language of the SPIN model checker [12], while in Section 5 we discuss what kind of properties can be checked in our current implementation. Finally in Section 6 we present an evaluation of our approach and a comparison with other works in the field of institutions and normative systems.

2 The Institution Metamodel

Many researches on institutions share several common or strongly related notions such as the concepts of role, norms, and authorization [4, 9, 10, 20]. In this section we introduce our institution metamodel to enhance a comparison with other works and to explain what are the key concepts that we perceive as essential to specify an institution, the relationships existing between them and their intended meaning.

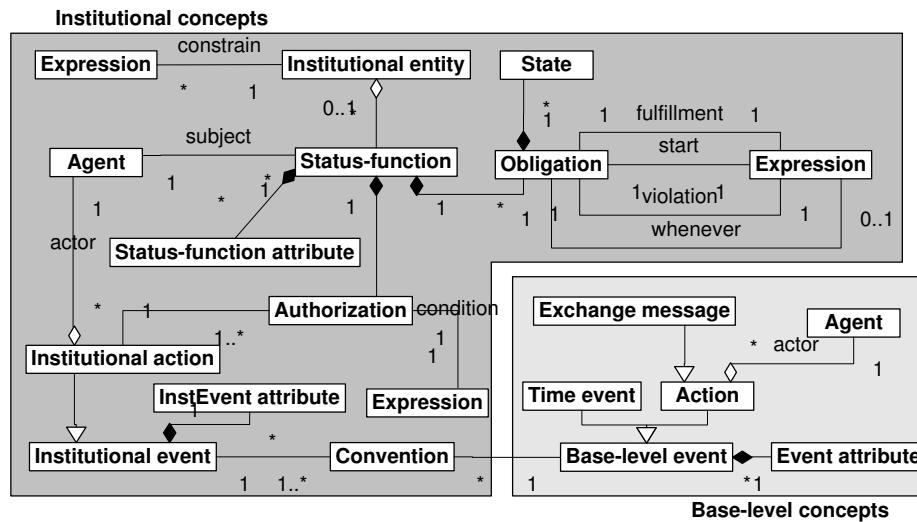


Fig. 1. The institution metamodel.

As shown in Figure 1, our metamodel is based on the notion of *agent status function*, that is, a status imposed on an agent and recognized as existing by a set of agents. Typical examples of status functions are not only the concept of auctioneer, participant, or winner of an auction, but also being the owner of a good, being the husband or the wife of somebody. The concept of status function shares several features with the concept of role as it has been discussed in the literature (refer to [1, 16] for an overview). Despite that, we prefer to use the term status function for three reason: (i) the term role has been used with different meanings and it has been characterized in terms of very different concepts such as mental states, tasks, duties, etc; (ii) the term status function better represents the fact that we are concerned with status assigned to agents to perform

several functions and whose existence depends on those agents that recognize them as existing; (iii) the concept of status function is broader than the concept of role as used, for example, in [1]. In fact, it seems to be difficult to describe in terms of a “preexisting organization” being the owner of a good or being under age, while it is quite natural to regard them as status functions imposed by a group of agents.

We define status functions as possibly empty aggregates of *deontic positions* that can be expressed in terms of two main concepts, *authorizations* (also named *institutionalized power* [14, 20]) and *obligations*. An obligation is characterized by a *state* and by four *expressions* (*whenever*, *start*, *fulfillment*, and *violation*) which are used to specify conditional obligations and when an obligation should be considered fulfilled or violated (more details can be found in [23]). An obligation is created because a status function is imposed, changes its state when its conditions are satisfied, and eventually reaches a final state (*fulfillment*, *violation*, *revoked*) either because its expressions are evaluated to true or because it is associated to a revoked status function. Obligations can be also used to express prohibitions by specifying suitable violation expression, while we do not define a specific construct to explicitly represent the fact that an agent is permitted to perform an action as in [4, 10, 20]. Instead, as in [24] we consider that every *authorized* action, if it is not prohibited, is also permitted.

Interdependent status functions are declared within *institutional entities* which enforce on a group of status functions a set of *constraints* (e.g. an auctioneer cannot be also a participant). Moreover, institutional entities define cardinality constraints, like “an auction is composed by an auctioneer and a set of participants”.

Base-level events reflect changes that are produced in the physical world or that are relative to lower level institutions, like *time* events and *exchange message* events. We do not name these kinds of events “physical” because, strictly speaking, most events are somehow dependent on language and therefore institutional [21]. Despite that, in most cases the deontic positions we want to model are not affected by the institutional nature of those events, and therefore we can ignore it. In FIEVeL base-level events are modelled as if they were perceived and analyzed by a centralized *institution manager*, which manages the state of the system and updates it when an event occurs. Although such an assumption would be unrealistic in the implementation of a distributed system, we introduce it to reduce the complexity of the verified model. Also, it can be noticed that several prototypes of institutions [6, 11] and normative systems [8] consider a single centralized component which makes the institutional (normative) state evolve.

In the literature only agent actions have been considered relevant to describe institutions [4, 9, 20, 22, 24], and the attention has been focused on a single action type, namely the act of exchanging a message [7]. In our approach we are also interested in modelling the institutional effects of events that are not generated by agents, like for instance time events. In particular, a time event can counts as an institutional event (e.g. in most cultures the 18th birthday imposes new status functions on a person) or can cause the evolution of the state of an obligation. Therefore, it could be the case that an environmental event leads the system from a legal state (where no violations have occurred) to an illegal state, while permitted actions executed in a legal state must lead only to a legal state as described in [22].

Institutional events are not directly produced by the environment or by an agent thanks to its own capabilities, because their effects need to be recognized by a set of agents. Instead, institutional events happen because agents accept that when certain base-level events occur, if certain contextual conditions are satisfied, they *count as* institutional events. Therefore, there exists an ontological difference among base-level events and institutional events: while the former exist because they correspond to certain physical changes, the latter exist because they are recognized as existing by a community of agents.

In [9] we discussed what kind of contextual conditions are relevant to model that an agent action counts as an institutional action. In particular, we single out the following conditions: (i) there must exist a convention, namely a message type, binding the agent action to the institutional action; (ii) the system must satisfy the preconditions associated to the institutional action; (iii) the agent must be authorized to perform the institutional action. In this paper we extend the treatment of the count-as relation by considering institutional events in general. Therefore, the act of exchanging a message cannot be taken as our unique primitive to specify conventions. Instead, any base-level event can be used to define a new convention. As in [9] an ontological precondition must be satisfied, although here preconditions are expressed in terms of the existence of absence of certain status functions. Finally, in the case of institutional actions, the agent must also be authorized. If all these conditions are satisfied, a base-level event counts as an institutional actions and its effects take place, which means that certain status functions will be assigned or revoked.

3 FIEVeL

Figure 1 not only depicts our metamodel of institutions, but it actually represents the abstract syntax of FIEVeL (*Functions for Institutionalized Environments Verification Language*), a language that has been defined to *model* institutions in terms of the concepts introduced by our metamodel and to *verify* them by applying model-checking techniques [2, 19]. For this reason, FIEVeL has been designed to limit the *state explosion problem* [2] and to be translated into Promela, the input language of the SPIN model checker [12]. In the following of this Section we will discuss the formal framework we have developed to define the semantics of FIEVeL and we will exemplify FIEVeL constructs by reporting few fragments extracted by our specification of the English Auction institution. More details about the formal semantics of FIEVeL, its grammar and the full specification of the English Auction can be found in [23].

A FIEVeL specification may be composed of several institutions that are concurrently executed, although at the moment FIEVeL does not allow the development of interdependent institutions, that is, institutions whose state and evolution directly depend on the state of another institution. A FIEVeL specification is composed by four main sections:

- *basic-domain definition*, where a designer can define new basic types by specifying them as subsets of built-in basic types or by enumerating their elements. FIEVeL defines few built-in basic types, like integers (*int*), agent identifiers (*AID*), objects

- identifiers (*OID*) and date stamps (*Date*). User defined basic types can then be used to define complex types (status functions, events, etc.).
- *base-events definitions*, where a designer can list all the relevant messages, constituted by a *message type* and a (possible empty) set of attributes, which can affect the institutions defined in the current specification. At the moment FIEVeL assumes that all messages belong to a single type of communicative acts, namely, *declarations* [9].
 - *institution definitions*, where for each institution it is possible to specify a set of institutional entities and status functions, a possibly empty set of constraints on them, and a set of conventions binding base-level events to the performance of several institutional-events. In Figure 2 it can be noticed that preconditions of institutional events are specified in terms of the existence or absence of certain status functions, while their effects are expressed by assigning (or revoking) several status functions on agents. As in [15], an institutional event describes some changes among two consecutive states representing which “variables” have been modified and their new values.
 - *model definition*, where elements composing each basic domain and the initial state of the system, described by imposing certain status functions on agents, are defined. Domain elements are specified because SPIN is able to verify only finite models.

In Figure 2 we report a few fragments of our specification of the English Auction institution, showing that an auctioneer that has received at least one offer (represented by OFFERED status function) is not only authorized to declare the current winner if no agent can still make a bid (INROUND), but since then it is also obliged to do so before the next time instant has elapsed. The action *currentWinner* is conventionally bound to the exchange of a message of type *decCurrentWinner* and, if the agent indicated as winner of the round is the first which has offered the highest price, therefore that agent is declared *current winner* and offers relative to the previous round are cancelled.

In FIEVeL time is considered as a component of the system and is explicitly represented with a variable whose value is incremented after each time event and that can be accessed by means of the *now* function. Two consecutive time events t_i and t_{i+1} may be separated by a sequence (possible empty) of other base level events, which are assumed to occur at time t_i . Hence the institutional state may change due, for instance, to message-exchange events even if the time variable does not change. Therefore, time aspects are regarded in two distinct ways: (i) as in classical temporal logic, to define *qualitative* properties (e.g. it is always the case that an auctioneer cannot win an auction), and (ii) as in RTTL [18] to express *quantitative* properties (e.g. the auctioneer must open the auction before two minutes since now).

Every FIEVeL specification corresponds to an *ideal* transition system characterized by a *many sorted first-order signature* where every transition represents all institutional effects (institutional actions, fulfillment of obligations, etc) associated with a base-level event. Actually, the generation of such transition system is not only computationally expensive, but it would also lead to a huge number of Promela code lines. In fact, due to several limitations of SPIN, for every institution instance we should generate a transition for each possible combination of base-level events, institutional-events, and obligation state change. For this reason, instead of calculating the ideal transition sys-

```

basic-types:
  priceD subtype-of int;
base-events :
  message decOffer(agent : AID, price : priceD, num : int);
  ...
institution EnglishAuctionInstitution {
  institutional-entity englishAuction {
  ...
  [0,ALL] status-function OFFERED (bidder: AID,
  price: priceD, num: int) {
  key: bidder;
  authorizations:
    currentWinner c <- not exists x in INROUND [true];
  deontic-specification:
    obligation(not exists x in INROUND[true],done(currentWinner),
    now()>activation-time+1);
  }// OFFERED
  ...
  constraints:
    AUCTIONEER disjoint PARTICIPANT;
  }// entity
  conventions:
    exch-Msg(decCurrentWinner (ag,pr,r)) =c=> currentWinner(ag,pr,r);
  ...
  institutional-events:
    institutional-action currentWinner(agent :AID,
    price :priceD ,n: int):
    pre: (exists o in OFFERED[((o.subject = actor and o.bidder=agent)
    and (o.price=price and o.num=n))] and forall x in OFFERED
    [(x.price < price or (x.price=price and n<=x.num))]);
    eff: p in PAR -X->
      assign(p.subject, CURWINNER(price))[p.subject = agent],
      o in OFF -X->
        revoke(o.subject, OFFERED(o.agent, o.price, o.n))[true];
    ...
  }// institution
model-definition:
  basic-domains:
    AID={aid_0,aid_1,aid_2};
    ...
  institutions:
    EnglishAuctionInstitution initial-state:
      assign(aid_0,AUCTIONEER());
    ...

```

Fig.2. Fragments of the English Auction institution specification.

tem, we prefer to derive a *computational* transition system, such that each transition partially represents the institutional effects of an event, so that summing the effects of a sequence of transitions we can reach the same institutional state.

To demonstrate that we can build a computational transition system which simulates the ideal system, that is, satisfies the same properties we want to verify, let $M = (S, S_0, R, V)$ be a Kripke structure over a set AP of atomic propositions such that S is a finite nonempty set of *states*, $S_0 \subseteq S$ is a finite nonempty set of *initial states*, R is a total relation on S , and V is a valuation function associating a value in $\{0, 1\}$ at each atomic proposition p in AP for each state. In the sequel we write π_k for the k -th state of path $\pi = s_0, s_1, s_2, \dots$ and π^k for the suffix of π starting at state π_k , that is, the sequence $s_k, s_{k+1}, s_{k+2}, \dots$. Π stands for the set of all paths in M , while Π_0 represents the set of all initialized paths in M , that is, the set of all paths whose first state is an initial state (formally: $\Pi_0 = \{\pi \in \Pi \mid \pi_0 \in S_0\}$).

Let \widehat{M} be a second Kripke structure defined over a set of atomic proposition \widehat{AP} such that $AP \subseteq \widehat{AP}$; we define a relation $Z \subseteq \Pi \times \widehat{\Pi}$ such that $(\pi, \widehat{\pi}) \in Z$ if and only if for each proposition $p \in AP$:

1. $V(\pi_0, p) = \widehat{V}(\widehat{\pi}_0, p)$;
2. there exists a $k > 0$ such that:
 - (a) $V(\pi_1, p) = \widehat{V}(\widehat{\pi}_k, p)$;
 - (b) for every $0 < r < k$ $V(\pi_0, p) = \widehat{V}(\widehat{\pi}_r, p)$;
 - (c) $(\pi^1, \widehat{\pi}^k) \in Z$.

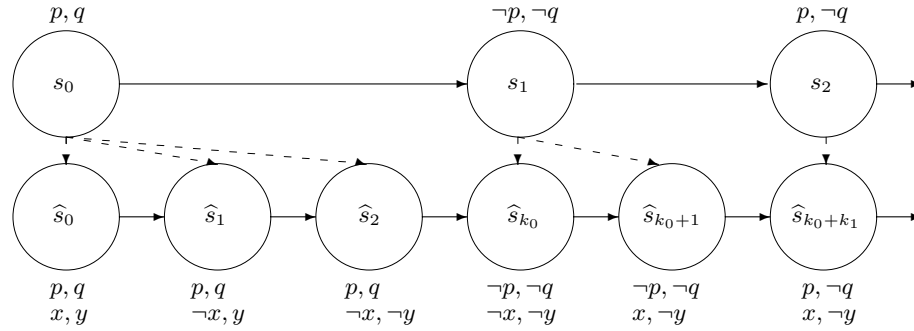


Fig. 3. Simulation relation among paths of different models with respect to propositions p and q . Dashed arrows represent the Z relation.

Let us consider a Kripke structure M characterized by a set $AP = \{p, q\}$ of atomic propositions and let \widehat{M} be a second Kripke structure defined over $\widehat{AP} = \{p, q, x, y\}$. Figure 3 depicts two paths π and $\widehat{\pi}$ related by the Z relation $((\pi, \widehat{\pi}) \in Z)$. The reader can observe that: (i) the valuation of common propositions p and q is the same in the first state; (ii) there exists a segment of length k_0 such that state $\widehat{\pi}_{k_0}$ has the same valuation

of π_1 and all other states of the segment have the same valuation of state π_0 ; the suffix $\widehat{\pi}^{k_0}$ is related by Z to the suffix of π starting from π_1 .

Path $\widehat{\pi}$ is therefore emulating the behavior of path π by simulating each transition on path π with a finite sequence of transitions, such that it keeps unchanged the truth values of common atomic propositions until it changes them in a single step. Intuitively, we can imagine building a path $\widehat{\pi}$ by taking path π and inserting between two consecutive states (π_i, π_{i+1}) a new set of intermediate states which are characterized by the same valuation function of state π_i .

It can be demonstrated (see [23]) that if for each $\pi \in \Pi_0$ exists a $\widehat{\pi} \in \widehat{\Pi}_0$ such that $(\pi, \widehat{\pi}) \in Z$, and for each $\widehat{\pi} \in \widehat{\Pi}_0$ exists a $\pi \in \Pi_0$ such that $(\pi, \widehat{\pi}) \in Z$, for each temporal formula φ composed by atomic propositions belonging to AP and which does not contain the *next* temporal operator, $M \models \varphi$ with respect to initialized paths Π_0 if and only if $\widehat{M} \models \varphi$ with respect to $\widehat{\Pi}_0$. This result can be extended to transition systems (see [23]) and it means that given an *ideal* transition system which corresponds to a FIEVeL specification, we can build a *computational* transition system which preserves all temporal properties of the *ideal* system. It can be observed that to obtain this result we have to renounce to the *next* operator, which does not represent a problem considering that not even SPIN supports it for efficiency reasons. Although in [5] the *next* temporal operator has been used to specify interaction protocols, we think that its use is problematic if we consider as possible events not only message exchanges but also time events, since it is not longer ensured that the next state is due to a message exchange. For this reason, we think that renouncing to the *next* operator in the specification of an institution is a sensible choice.

Thanks to this result, in [23] we define the semantics of FIEVeL in terms of a computational transition system, although, at a more abstract level, the designer may reason in terms of the ideal transition system. Moreover, it guarantees that FIEVeL specifications can be verified with SPIN and that every property regarding the institutional state holding in the ideal institution also holds in the Promela model, and vice versa.

4 Translating FIEVeL into Promela

A Promela specification is composed by a set of *processes* and *global variables* that can be described by defining new *process types* and *record structures*. Essentially a Promela process is constituted by a set of statements, which can be *simple statements*, like assignments, or *compound statements*, like selection (`if`) and repetition (`do`). Each statement is characterized by an *enabling condition* and a *postcondition*. Promela imposes severe restrictions on what can be specified in a precondition, therefore, to overcome such limitations and to increase the expressiveness of FIEVeL without producing an huge number of intermediate states, we chose to use embedded C code to calculate transition preconditions and their effects. The SPIN model checker adopts an interleaving semantics, which means that when several processes have executable statements, it randomly chooses one of them and executes it. When all enabling conditions are evaluated to false, two special preconditions (also named guards), `timeout` and `else` are evaluated to *true*. In particular, `else` is enabled only if all transitions at the process level cannot be executed, while `timeout` is evaluated to true only if no process has an

enabled transition. In this brief overview we have just introduced a few concepts that are necessary for the sake of the present discussion, while further details can be found in [12].

To model check FIEVeL specifications we have defined an automatic translation of FIEVeL into Promela which proceeds as follows. First, status functions, institutional entities, obligations, base events and institutional events are translated into a set of type definitions, which are then used to declare a set of variables representing the state of the system. According to the framework discussed in Section 3, in Promela we identify two groups of variables, one representing the *current institutional state* and another which is exploited to generate the *next institutional state*.

Each institution contained in a FIEVeL specification is then translated into a new process definition according to the pattern represented in Figure 4. Each process instance is associated to an identifier to bind the process with an appropriate set of variables representing its state. Every process representing an institution consists of a main loop which is enabled whenever a new base event is generated. For the sake of simplicity, we can imagine that the process contains an inner loop where each option represents an institutional event or an obligation state transition. Actually, to further reduce the number of intermediate transitions and to generate a more compact Promela code, our current implementation introduces several improvements that have not been reported in Figure 4.

```

proctype institutionProc(int id) {
  do
  :: (nextEvent.analyzed[id]==FALSE)->
  do
  :: (condition_inst_event_x1 || condition_inst_event_x2)->
    apply_effects(next_event);
  ...
  :: (condition_inst_event_y1 || condition_inst_event_y2)->
    apply_effects(next_event);
  :: ((next_obligation.state==inactive) &&
    (start_obligation && obligation.state==inactive))->
    next_obligation.state=active;
  :: else -> break;
  od
  nextEvent.analyzed[id]=TRUE;
od
}

```

Fig. 4. Translation pattern for processes representing institutions.

As discussed in Sections 2 and 3, institutional states evolve because base-level events happen and they count-as institutional events. To model base level events, we define a new process, named *eventGenerator*, which generates actions and events as if they were produced by agents or the environment. Actually, since we are concerned with the analysis and verification of institutional reality, at each stage we only consider those events that, according to the institutional state of the system, can count as an institutional event and we randomly generate one of them. The general structure of an *eventGenerator* is represented in Figure 5.

Finally we declare an *init* process to generate the initial state of the system according to the model-definition section of a FIEVeL specification. Essentially, the *init* process sets the initial values of the variables and for each institution instance creates a process of the corresponding process type.

To intuitively show that our translation respects the framework depicted in Section 3, we can observe that the *eventGenerator* is executed only when no other processes have an enabled statement, as guaranteed by the `timeout` guard. Since it is activated, the *eventGenerator* updates the *current state* with the values contained in the *next state* variables, calculates the truth value of a set of propositions representing counts-as conditions (preconditions and authorizations) and finally, if it is possible and if a *termination condition* has not been reached, generates a new event. Finally, it sets the next base-level event, which also means that *analyzed* variables are set to *false*, and since then all other processes can be executed. Therefore, according to the process structures presented in Figure 4 and Figure 5, we ensure that whenever an *eventGenerator* is executed, any other process cannot be interleaved. Also, we guarantee that the *eventGenerator* is executed only when all institutions instances have generated the next institutional state by considering all possible institutional transitions (counts-as, institutional-event effects, and obligation transitions), such that current state variables can be updated with the new calculated values.

```

proctype eventGenerator(){
  do
    :: timeout ->
      curr_state=next_state;
      updateConditions();
      if
        :: !(end_condition)->
          if
            :: (true) -> next_event = timer; next_time++;
            :: condition_event1 -> next_event = event1;
            :: condition_event2 -> next_event = event2;
            ...
            :: condition_eventn -> next_event = eventn;
          fi
        fi
      od
  }

```

Fig. 5. The structure of an eventGenerator process.

An important aspect of our translation of FIEVeL into Promela is that we can directly check LTL properties, instead of writing assertions embedded in the Promela code. As a consequence, properties definition results significantly simplified and it can be automated, although at the moment this feature has not been implemented yet in our tool.

5 Verifying Institutions

Once an institution has been defined with FIEVeL and translated into Promela, we think that there are two main type of properties that must be verified, *soundness properties* and *domain specific properties*. Soundness properties represent general desirable properties of institutions which stem from the metamodel: for example, the soundness of an institution specification requires the consistency of the set of deontic-powers associated to a status function (*weak consistency*) and the consistency of all the obligation of status functions that may be assigned to an agent (*strong consistency*). Instead, domain specific properties stem from peculiar features of the specified model and regard the functionality of an institution: for instance, we may check if it is possible that a participant is declared to be the winner of an auction. In the following we exemplify our approach by defining several LTL properties that can be checked in our approach and we show how they can be verified in SPIN.

Soundness properties are often defined to guarantee the rationality of an institution. For instance, an institutional event must eventually happen in at least one execution. By checking the *satisfiability* of the following formula, where we combine FIEVeL expressions with temporal operators, we can verify that an event of type *current_winner_is* must happen in least one execution:

$$\mathbf{F}happens(current_winner_is)$$

Temporal operators and FIEVeL expressions can be combined because we assume that semantics domains are fixed at all states of our system. The SPIN model checker only admits propositional symbols into LTL formulae [12], hence we must introduce new propositional symbols corresponding to FIEVeL expressions. For instance, the *happens* expression reported above is translated as follows:

```
#define happens_currentWinner (lastEvent.EAI_currentWinner==true)
```

If the property does not hold, the model checker returns a counterexample which shows a possible trace that violates it. To optimize the verification process we have encoded several controls into the FIEVeL translator, e.g. the specification defines at least one convention for each institutional event.

To conclude this section we report an example of a domain specific property that allows us to verify that desired institutional states may be reached, which implies that our specification defines all the institutional actions and authorizations needed for the correct evolution of the system. For instance, we may want to check that only a participant can become the winner of an auction. This requirement also means that an auctioneer cannot win an auction and that any participant may eventually become the winner. These properties are formalized as follows:

$$\mathbf{G}\neg(auctioneer \wedge winner) \tag{1}$$

$$\mathbf{F}(participant \wedge winner) \tag{2}$$

By verifying if property 1 is *valid* and if property 2 is *satisfiable* and by analyzing the generated counterexamples, we discovered and fixed an error in an earlier version

of our specification of the English Auction. In fact, the effects of the institutional action *current_winner.is* were accidentally specified such that the status function of current-winner was assigned to the actor of that action, instead of to the agent declared to be the winner (see Section 3). Clearly, given that such action is authorized only for the auctioneer, in that specification only an auctioneer could win an auction.

To provide the reader with a feeling of the computational costs of our approach, we report results obtained during the verification of property 1 on a desktop PC with installed Windows and equipped with a pentium 3.0 GHz and 512 MB of RAM. Clearly, results reported in Table 1 are severely influenced by the small amount of memory we had at our disposal. Despite that, we consider our results interesting, considered the complexity of the specification and compared with our experience in the verification of systems specified in Promela.

Table 1 reports our experiments, where “HCm” indicates that we have applied the Hash Compression mode while “*” represents the fact that the verification process requires more than 500 seconds and therefore it has been interrupted. Analyzing results showed in Table 1, we can observe that the number of agents and prices, but especially the number of possible time events contribute to a very fast growth of time and memory required to verify the property. This is essentially due to two different factors, one relative to our framework and one which depends on the chosen example. The representation of time as an explicit variable of the system somehow introduces an explicit counter, which leads the model checker to consider as different similar states which just differ in the value of the clock. In our formalization of the English Auction the order in which agents make their bids is associated to each OFFERED status function (see Section 3), so that the auctioneer is able to declare a *current winner* even if two agents have offered the same price during a round (the first agent that has bid the highest price is declared to be the current winner). Clearly, keeping trace of the relative order of offers contributes to increasing the number of possible paths that the model checker has to consider, and hence the amount of required memory and time.

6 Discussion and Conclusions

In this paper we have presented a framework for verifying institutional reality based on the notion of status function, which is regarded as a possibly empty aggregate of authorizations and obligations. Our approach is motivated by the fact that institutional events modify institutional facts and, as a consequence, agents authorizations and obligations are changed. For this reason, we chose to characterize institutional events in terms of imposition and revocation of agent status functions, which are taken as our basic concept. We have also introduced FIEVeL, a language which allows to specify artificial institutions in terms of the concepts of our metamodel. We have also briefly discussed how FIEVeL specifications can be translated into Promela, such that soundness properties can be defined and model-checked with SPIN.

Several attempts have been previously carried out to apply model checking techniques and tools to verify *electronic institutions*, a formalism proposed in [7] to specify institutions. Roughly speaking, the language defined in [7] describes institutions as finite automata, and starting from this point [13] and [3] focus their attention on

| Agents | Time events | Prices | Memory (MB) | Time (sec) | HCm memory (MB) | HCm time (sec) |
|--------|-------------|--------|-------------|------------|-----------------|----------------|
| 3 | 2 | 5 | 94.71 | 4 | 4.96 | 2 |
| 3 | 2 | 6 | 155.78 | 7 | 6.60 | 4 |
| 3 | 2 | 7 | 252.90 | 12 | 8.65 | 8 |
| 4 | 2 | 5 | 150.03 | 10 | 5.79 | 6 |
| 4 | 2 | 6 | 257.40 | 18 | 7.43 | 14 |
| 4 | 2 | 7 | 421.32 | 123 | 9.47 | 23 |
| 3 | 3 | 5 | 309.85 | 16 | 9.47 | 10 |
| 3 | 3 | 6 | 511.60 | 374 | 13.98 | 13 |
| 3 | 3 | 7 | * | * | 20.53 | 25 |
| 4 | 3 | 5 | 505.60 | 165 | 10.70 | 23 |
| 4 | 3 | 6 | * | * | 15.62 | 43 |
| 4 | 3 | 7 | * | * | 23.00 | 56 |
| 4 | 4 | 7 | * | * | 52.10 | 147 |
| 4 | 5 | 7 | * | * | 109.05 | 444 |

Table 1. Time and memory required to verify $\mathbf{G}\neg(\text{auctioneer} \wedge \text{winner})$. Results are reported showing the size of the considered models.

properties of finite automata (e.g. “it is always possible to reach a final state”), while they do not take into account norms. Instead, in our approach the attention is essentially focused on verifying properties of institutional states, described in terms of status functions, which are intrinsically related with norms of the system and thereby on checking whether norms and authorizations defined by the designer are consistent. In this respect, our approach is more similar to the one presented in [4], where answer set programming techniques are applied to represent and analyze institutions. In [4] the authors distinguish among *institutional domain facts* and *normative facts* (obligations, institutionalized power, and permissions), while in this paper we proposed a unified view of institutional domain and normative facts. Indeed, we claim that institutional facts are such only because they imply new normative facts for the interacting agents, which also represents a significant difference with respect to our previous attempts to model institutional reality [9, 24].

As [4, 24], in [20] several key concepts like authorizations, obligations and prohibition are used to model institutional reality. It can be observed that the formalism used in [20] to specify institutions (named *normative systems*) does not provide any abstraction to describe that every institutional action must be authorized in order to be successfully executed. Instead, the authors must specify this fact for every single action and for every role defined by the institution. Therefore, the definition of a metamodel provides a significant advantage, especially when many status function (or roles, using the terminology of [20]) are authorized to perform the same institutional action. Furthermore, the definition of an automatic translation of FIEVeL into Promela allows us to verify our specification, while in [20] the authors must rely on “systematic runs”.

At the moment we have started to implement a simulator of specifications written in FIEVeL, which should help the designer to test his own specifications and to understand how they will evolve by interactively determining events that will be processed by the

institution. Furthermore, we plan to use the very same simulator to execute the counterexamples generated by SPIN to help the designer to reason in terms of the abstractions provided by FIEVeL and to hide the details related with Promela and its syntax. Finally we will extend our metamodel, and consequently FIEVeL, to model different interdependent institutions like in [24], which raises, among others, two interesting research problems: first, how to model interdependencies among different contexts, and second, how to design an institution which somehow depends on another institution.

References

1. G. Boella and L. van der Torre. The ontological properties of social roles: Definitional dependence, powers and roles playing roles. In *Proceedings of the ICAIL05 Workshop on Legal Ontologies and Artificial Intelligence Techniques*, 2005.
2. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.
3. O. Cliffe and J. Padget. A Framework For Checking Interactions Within Agent Institutions. In *Proceedings of the ECAI Workshop on Model Checking and Artificial Intelligence (MoChart-2002)*, 2002.
4. O. Cliffe, M. D. Vos, and J. Padget. Specifying and Analysing Agent-based Social Institutions using Answer Set Programming. In D. Lindermann, S. Ossowski, J. Padget, and J. Vázquez-Salceda, editors, *Proceedings of the AAMAS05 Workshop on Agents, Norms and Institutions for Regulated Multiagent Systems*, pages 149–162, 2005.
5. U. Endriss. Temporal logics for representing agent communication protocols. In *Proceedings of the AAMAS Workshop on Agent Communication (AC-2005)*. Springer-Verlag, 2005. To appear.
6. M. Esteva, J. A. Rodríguez-Aguilar, B. Rosell, and J. L. Arcos. AMELI: An Agent-based Middleware for Electronic Institutions. In N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, pages 236–243. ACM Press, 2004.
7. M. Esteva, J. A. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specification of electronic institutions. In F. Dignum and C. Sierra, editors, *Agent Mediated Electronic Commerce, The European AgentLink Perspective*, volume 1991 of *LNAI*, pages 126–147. Springer, 2001.
8. A. D. H. Farrell, M. J. Sergot, M. Sallé, and C. Bartolini. Using the Event Calculus for Tracking the Normative State of Contracts. *Journal of Cooperative Information Systems*, 14(2-3):99–129, 2005.
9. N. Fornara, F. Viganò, and M. Colombetti. Agent communication and institutional reality. In R. van Eijk, M. Huget, and F. Dignum, editors, *Agent Communication*, volume 3396 of *LNAI*, pages 1–17. Springer Verlag, 2005.
10. A. García-Camino, P. Noriega, and J. A. Rodríguez-Aguilar. Implementing norms in electronic institutions. In *Proceedings of the 4th International Joint Conference on Autonomous agents and Multi-Agent Systems (AAMAS 2005)*, pages 667–673. ACM Press, 2005.
11. A. García-Camino, J. A. Rodríguez-Aguilar, C. Sierra, and W. W. Vasconcelos. A distributed architecture for norm-aware agent societies. In M. Baldoni, U. Endriss, A. Omicini, and P. Torroni, editors, *Declarative Agent Languages and Technologies III, Third International Workshop, DALT 2005, Utrecht, The Netherlands, July 25, 2005, Selected and Revised Papers*, volume 3904 of *LNCS*, pages 89–105. Springer, 2005.
12. G. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison Wesley, 2003.

13. M.-P. Huget, M. Esteva, S. Phelps, C. Sierra, and M. Wooldridge. Model Checking Electronic Institutions. In *Proceedings of the ECAI Workshop on Model Checking and Artificial Intelligence (MoChart-2002)*, 2002.
14. A. Jones and M. J. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, 1996.
15. R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.
16. C. Masolo, L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi, and N. Guarino. Social Roles and their Descriptions . In D. Dubois, C. Welty, and M. Williams, editors, *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004)*, pages 267–277, 2004.
17. D. North. *Institutions, Institutional Change and Economics Performance*. Cambridge University Press, Cambridge, United Kingdom, 1990.
18. J. S. Ostroff. Modelling, specifying and verifying real-time temporal interval logic. In *Proceedings of the IEEE Symposium on Real-Time Systems.*, New York, 1987. IEEE.
19. D. A. Peled. *Software reliability methods*. Texts in Computer Science. Springer-Verlag New York, Inc., New York, NJ USA, 2001.
20. J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Formalization of a voting protocol for virtual organizations. In *Proceedings of the 4th International Joint Conference on Autonomous agents and Multi-Agent Systems (AAMAS 2005)*, pages 373–380, New York, NY, USA, 2005. ACM Press.
21. J. R. Searle. *The construction of social reality*. Free Press, New York, USA, 1995.
22. M. J. Sergot. Modelling unreliable and untrustworthy agent behaviour. In B. Dunin-Keplicz and A. Jankowski and A. Skowron and M. Szczuka, editor, *Monitoring, Security, and Rescue Techniques in Multiagent Systems*, Advances in Soft Computing, pages 161–178. Springer-Verlag, 2005.
23. F. Viganò. FIEVeL, a language for the specification and verification of artificial institutions. Technical Report 3, Institute for Communication Technologies, Università della Svizzera Italiana, 2006. To appear.
24. F. Viganò, N. Fornara, and M. Colombetti. An Event Driven Approach to Norms in Artificial Institutions. In D. Lindermann, S. Ossowski, J. Padget, and J. Vázquez-Salceda, editors, *Proceedings of the AAMAS05 Workshop on Agents, Norms and Institutions for Regulated Multiagent Systems*, pages 149–162, 2005.