Institute for Communication Technologies (ITC)
www.itc.com.unisi.ch

Faculty of Communication Sciences

Università della Svizzera italiana
via Buffi 13, CH-6900 Lugano, Switzerland

Technical report No. 5, 2008

Formal Specification of Artificial Institutions Using the Event Calculus

Nicoletta Fornara and Marco Colombetti

Formal Specification of Artificial Institutions Using the Event Calculus*

Nicoletta Fornara[1] and Marco Colombetti[1,2]
[1]Università della Svizzera italiana,
Via G. Buffi 13, 6900 Lugano, Switzerland
[2]Politecnico di Milano,
Piazza Leonardo Da Vinci 32, Milano, Italy

Abstract

The specification of open interaction systems, which may be dynamically entered and left by autonomous agents, is widely recognized to be a crucial issue in the development of distributed applications on the internet. The specification of such systems involves two main problems: the first is the definition of a standard way of specifying a communication language for the interacting agents and the context of the interaction; the second, which derives from the assumption of the agents' autonomy, is finding a way to regulate interactions so that agents may have reliable expectations on the future development of the system. A possible approach to solve those problems consists in modelling the interaction systems as a set of artificial institutions. In this chapter we address this issue by formally defining, in the Event Calculus, a repertoire of abstract concepts (like commitment, institutional power, role, norm) that can be used to specify artificial institutions. We then show how, starting from the formal specification of a system and using a suitable tool, it is possible to simulate and monitor the systems evolution through automatic deduction.

1 Introduction

The specification of *open interaction systems*, which can be *dynamically* entered and left by *heterogeneous*, *autonomous*, and *self-interested* agents, is widely recognized to be a crucial issue in the development of distributed applications on the Internet. The interacting agents may be heterogeneous because they may be developed by different designers; as a consequence, no assumptions can be made on their internal architecture and it is impossible to access their internal states. Agents may be self-interested because they act on behalf of different human counterparts that, in general, do not share a common goal. Finally,

*Supported by the Hasler Foundation, Project 2204, "Artificial institutions: Specification of open distributed interaction systems"

agents are autonomous because they are not centrally controlled, but act on the basis of private strategies.

The specification of such systems involves two main problems: the first is the definition of a standard way of specifying a communication language for the interacting agents and of the context of the interaction; the second, which derives from the assumption of the agents autonomy, is finding a way to regulate interactions so that agents may have reliable expectations on the future development of the system. A possible approach to solve those problems consists in modelling the interaction systems as a set of *artificial institutions*. By this term we mean the digital counterpart or extension of a human institution, like for example the institution of language, the institution of property, or the institution of auctions.

In our view the definition of a specific artificial institution [20] consists of: (i) a component, called meta-model, which includes the definition of basic entities common to the specification of every institution, like the concepts of commitment, institutional power, role, and norm, and the actions necessary for exchanging messages; (ii) a component specific to the institution in question, which includes the specification of the powers and norms that apply to the agents playing roles in the institution, and the definition of the concepts pertaining to the domain of the interaction (for example the actions of paying or delivering a product, bidding in an auction, etc.). The aim of this chapter is to give a formal definition of the domain-independent component and to illustrate the domain-dependent component through a meaningful example.

In the literature on multi-agent systems, various concepts that have some similarities or analogies with our idea of an artificial institution have been proposed. In the Nineteen-eighties, Carl Hewitt introduced the concept of *open systems* [24]; in [3] Artikis and colleagues give a detailed definition of the term *open societies* and use the term *institution* as a synonym, without distinguishing, as we do, an open system from an artificial institution; Noriega, Sierra and colleagues use the term *electronic institution* [32, 12] to indicate "the rules of the game-a set of conventions that articulate agents' interactions." According to Arcos and colleagues "The essential roles [electronic institutions] play are both descriptive and prescriptive: the institution makes the conventions explicit to participants, and it warrants their compliance." [2, pag 193]; therefore electronic institutions are very similar to our artificial institutions, even if usually they do not allow for the violation of rules and thus do not address the management of sanctions. In [39] Vázquez-Salceda and colleagues concentrate their attention on the *organizational* perspective of an open system. There are moreover works that are mainly focused on the normative component of a system, usually called *Normative Framework* [27] or *Normative System* [5].

In this chapter we formalize *OCeAN* (Ontology, CommitmEnts, Authorizations, Norms) [19, 20], a meta-model for the specification of artificial institutions, using a variant of the *Event Calculus*, the *Discrete Event Calculus* (DEC) introduced by Mueller [30]. Based on many-sorted first order logic, DEC is suited for reasoning about action and change: by means of axioms it is possible to describe how certain properties of a domain change according to the occurrence

of events or actions at different time points. The DEC formalization of *OCeAN* consists of a set of events, fluents, and axioms that describe the entities, and their evolution in time, that can be used in the specification of a wide range of artificial institutions. The specification of an interaction system is therefore given by the conjunction of those axioms with some domain-dependent definition of fluents, events, and axioms, plus the definition of the initial state of the system.

A fundamental advantage of using the Event Calculus for the specification of systems is that, exploiting suitable tools, it is possible to perform different types of reasoning, and in particular: *prediction*, where given an initial situation and a sequence of events the resulting situation is deduced; *planning*, where the sequence of actions that bring from an initial situation to a goal situation is discovered; and *postdiction*, where given a final state and a sequence of events one seeks an initial state. In this work we will mainly concentrate *deduction*, that is: given a description of a system based on the fundamental concepts of our meta-model, an *initial situation*, and an *event narrative* (i.e., a sequence of events that happens in the system), we are interested in tracking the evolution of the state of the system for simulation and monitoring purposes. To this aim, in Section 4 we use the Discrete Event Calculus Reasoner 1.0[1] [30, Chapter 13], a tool that relies on various satisfiability solvers and provers to perform reasoning.

The main contribution of this chapter with respect to our previous works [16, 19, 20, 17] is the formal definition of the *OCeAN* meta-model using the Discrete Event Calculus. The adoption a formal language for the specification of our meta-model leads us to revise and spell out in details certain concepts. In particular we have studied how to define powers at design time in terms of the roles played by the agents, and to dynamically let agents assign and remove powers during run-time. We have also refined our notion of role considering the possibility to define roles related to different institutional entities and the need to assign and dismiss roles during the dynamic evolution of the system. Finally we have revised mechanisms for the enforcement of norms through a detailed management of sanctions.

This chapter is organized as follows. In Section 2 we compare our approach with other existing proposals. In Section 3 we present our meta-model of artificial institutions. In Section 4 we give some examples of the formalization of a system and describe the simulation of their temporal evolution. Finally in Section 5 we draw some conclusions and discuss some possible directions for future research.

2 Background

In the literature there are some contributions using the Event Calculus for the formalization of various concepts that are relevant for agent interactions in open

[1] http://decreasoner.sourceforge.net/

systems. Yolum and Singh in [41] concentrate their attention on the formal specification of social commitment for the flexible specification of protocols. They reason about a given specification using Shanahan's abductive Event Calculus planner [36]. Their notion of commitment is quite similar to ours, even if their specification of the content (or condition) of commitments is context-dependent, and they do not have a notion of temporal proposition to express the interval of time when an action has to be (or not to be) performed.

Artikis et al. in [3] study the specification and animation of *open computational systems* using the Event Calculus. The software platform presented for automated animation of the global state of the system is very interesting. Their fundamental concepts (like power, normative position, and role) are close to the ones introduced in our work; the main difference is that in their work these concepts and the corresponding axioms are limited to the example proposed (the formalization of the Contract Net Protocol [38]), whereas in our work we present a set of concepts, and the axioms that characterize them, with the aim of defining an application-independent meta-model for the definition of open interaction systems.

In [13] Farrell and colleagues use an XML version of the Event Calculus called *ecXML* to specify *contracts*, for example a mail service agreement between a Service Provider and a Service Customer. This work highlights the importance of simulating and monitoring the evolution of contracts by means of a Java implementation of a reasoner. However, our use of the Discrete Event Calculus has the advantage of allowing for a richer variety of types of reasoning (e.g., planning), thanks to the existence of tools like the Discrete Event Calculus Reasoner 1.0. Moreover, like in the previous case our model is more general, and would treat contracts as a special type of artificial institutions.

Another interesting declarative approach to the definition of an Agent Communication Language, called *SC-IFF* [1], is based on the Social Integrity Constraints language. It is based on the notion of *expectation* (considered similar to a commitment) and on the idea of defining the semantics of communicative acts by specifying the future desirable communicative acts that have to be performed, within a certain instant of time, after the performance of a given communicative act; this without an explicit representation of the context of the interaction that in our model is obtained with the definition of various types of fluents, making the content of the acts simpler. The verification of one specification is possible thanks to a proof-procedure based on Abductive Logic Programming.

In the literature there are significant contributions on the specification of open multiagent systems adopting the institutional or organizational paradigm. The main difference between our approach and other proposals [3], [39], [2], [27], [21], [8], [7] is its completeness: we define a set of concepts and an Agent Communication Language (ACL) (inclusive of declarations) that if adopted as a standard would make it possible for an agent to interact with different systems without being redesigned. The semantics of communicative acts and of norms are both based on a notion of *social commitment* that is objective and external - two fundamental characteristics in systems where the internal architecture of the interacting agents is unknown and no assumption can be made on their

collaborative or competitive behavior. The systematic use of commitment has also the advantage of reducing the number of different constructs on which agents have to reason when planning their actions.

Among the previously mentioned articles the one which is closely related to our work is [39] where the *OMNI* framework for modelling agent organizations is presented. Given that they adopt an organizational perspective their model allows to specify the global goals of the system independently from the participating agents, similarly to our proposal they tackle the problem of specifying norms, whereas a crucial distinction that is not highlighted is the one between the power and the permission to perform an action, a crucial distinction when the semantics of declarations is defined.

3 The OCeAN meta-model

In our previous works [16, 19, 20, 17] we have presented and thoroughly discussed the *OCeAN* meta-model of artificial institutions that can be used to specify at a high level open interaction systems, where heterogeneous and autonomous agents may interact. In our view the fundamental components that characterize artificial institutions are: a *core ontology* for the definition of a set of concepts, actions/events, and *institutional actions*, used in the interaction; the *counts-as* relation and the notion of *power*, that are necessary for the concrete performance of institutional actions; and the *norm* construct, used to impose obligations and prohibitions to perform certain actions on agents interacting with the system, that are crucial for the specification of flexible interaction protocols, like for example the ones used in electronic auctions [40]. In particular powers and norms are expressed at design time in terms of the roles played by the agents. Such concepts, and in particular the fundamental notion of *commitment*, are used in Section 3.7 to specify an Agent Communication Language (ACL). Moreover to represent the *content* and the *condition* of commitments and norms, and their relation with time, we will introduce the notions of a *temporal proposition* and of an *evaluated temporal proposition*. Summarizing, in this chapter we introduce: the sorts *agent, action, and institutional action (iaction)*, which are fundamental for our meta-model; the following fluents for describing basic concepts:

- *Temporal Proposition (TP)* and *Evaluated Temporal Proposition (ETP)*;
- *Commitment* and *Precommitment*;
- *Power* and *Context*;
- *RoleOf* and *HasRole*;
- *Norm*;

and the events/actions that impact on such fluents, described by means of axioms. We will also introduce an action for exchanging messages and define a set of axioms that specify an Agent Communication Language (ACL).

3.1 Specification of an interaction system

In our view an open dynamic interaction system is a system that agents may use to interact with each other by means of a set of communicative acts. Those interactions may be in particular regulated by a set of norms that prescribe a flexible interaction protocol. The specification of an open dynamic interaction system (as exemplified in Section 4) consists of: (i) the set of fluents, events/actions and axioms that compose the *OCeAN* meta-model and will be presented in Section 3; (ii) those fluents, events/actions and axioms necessary for the specification of a category of artificial institutions (e.g., for the specification of the institution of auctions we have to introduce a fluent for representing auctions and axioms to create suitable powers and norms); (iii) those fluents, events/actions and axioms specific to a certain subtype of the category of institutions of interest (e.g., the English or the Dutch Auctions [19, 40]; (iv) when it is the case, the definition of artificial multi-institutions [9] that *use* other artificial institutions (the analysis of the definition of an institution using other institutions in important, but is beyond the scope of this chapter); (v) finally those fluents, events/actions and axioms specific to a concrete system (like those necessary to represent the products on sale and to specify the initial state of the system). To simulate the evolution of the system it is also necessary to define the history of the events that happen in the system at every instant of time.

3.2 The Formalism

3.2.1 Event Calculus

The formalism that we use to define the main concepts of an artificial institution is a version of the Event Calculus: the Discrete Event Calculus (DEC). The Event Calculus [26, 29, 30, 34, 35] is a formalism for reasoning about action and change that has been introduced by Kowalski and Sergot in 1986 and since then has evolved considerably [31]. In this work we use a version of the Event Calculus, the Discrete Event Calculus (DEC), introduced by Mueller [30] to improve the efficiency of automated reasoning by limiting time to the integers. DEC is adopted in the Discrete Event Calculus Reasoner 1.0, a tool that we used to test the axioms of our model of artificial institutions and to run a simulation of the specification of a system, as discussed in Section 4.

3.2.2 Conventions

The Event Calculus is based on many-sorted first order logic [30]. To make formulas more readable, we adopted a number of notational conventions.

Throughout the paper predicate symbols, function symbols and nonnumeric constants start with an uppercase letter, and variables start with lowercase letters. Variables that are not explicitly quantified are assumed to be universally quantified. The notation $\psi \equiv_{def} \varphi$ defines ψ as an abbreviation of φ. We adopt the unique name axioms [30, pag 31], whose meaning is that different constants refer to different objects.

If a sort is a subsort of another sort we separate the child sort from the parent sort with a colon; for example, *agent:object* means that *agent* is subsort of the sort *object*. In general, the sort of a variable is determined by removing the final digits from the variable; for example, the variable *agent1* has sort *agent*. When an axiom contains only one variable of a given sort, the name of the variable coincides with the name of the sort (i.e., the variable *agent* has sort *agent*). Sometimes, to have a more perspicuous variable name we do not follow this convention; in such cases the sort of the variable is specified in the text before the axiom using colon as separator, for example *debtor:agent*.

The sort of function symbols, including fluents, and of their arguments is specified through prototypes: the prototype $s_0\ f(s_1, ..., s_n)$, where $s_0, s_1, ..., s_n$ are sorts, means that function symbol f is to be applied to n terms of sorts $s_1, ..., s_n$, and that the resulting term is of sort s_0.

3.2.3 Discrete Event Calculus

The predicates of the Event Calculus are used "for saying what happens when, for describing the initial situation, for describing the effects of actions, and for saying what fluents hold at what times" [35]. The Event Calculus introduces the basic sorts *event*, *fluent*, and *timepoint* with variables e, f, and t. The predicates of the Event Calculus that will be used in this paper are:

- $Happens(e, t)$: event e happens at timepoint t. In the Event Calculus the performance of an action is regarded as an event. We assume that two events of the same type never happen at the same time instant; therefore an event type plus an instant of time univocally identify an event token.
- $HoldsAt(f, t)$: the fact described by fluent f holds at timepoint t.
- $Initiates(e, f, t)$: event e initiates fluent f at t. Its intuitive meaning is that if event/action e occurs at t then f will holds after t. Note that at a different time instant the occurrence of the same event may not start the fluent f.
- $Terminates(e, f, t)$: event e terminates fluent f at timepoint t. If e occurs at time t then f will no longer hold after t.

The $ReleasedAt(f, t)$ predicate, used in the following DEC axioms, means that fluent f is released from the commonsense law of inertia at time point t. The axioms of the Discrete Event Calculus (DEC) that are crucial for the comprehension of this paper are as follows (for a complete list see [30, pag 27]).

DEC5 $(HoldsAt(f, t) \wedge \neg ReleasedAt(f, t + 1) \wedge$
$\neg\exists e(Happens(e, t) \wedge Terminates(e, f, t))) \rightarrow HoldsAt(f, t + 1)$

DEC6 $(\neg HoldsAt(f, t) \wedge \neg ReleasedAt(f, t + 1) \wedge$
$\neg\exists e(Happens(e, t) \wedge Initiates(e, f, t))) \rightarrow \neg HoldsAt(f, t + 1)$

DEC9 $(Happens(e, t) \wedge Initiates(e, f, t)) \rightarrow HoldsAt(f, t + 1)$

DEC10 $(Happens(e, t) \wedge Terminates(e, f, t)) \rightarrow \neg HoldsAt(f, t + 1)$

3.3 Fundamental sorts

To the basic sorts of the Event Calculus we add the following sorts: *agent*, to represent the agents that interact with the system; *action:event*, a subsort of *event*, to represent those events that are brought about by agents, for example we may need to introduce the action for delivering a product: *action Deliver(agent,agent,product)*; *iaction:event*, a subsort of *event*, to represent *institutional actions*, that is, actions that change institutional attributes, which exist only thanks to common agreement. Note that agents cannot perform institutional actions by exploiting causal links occurring in the natural world, as would be done to open a door; rather, as we shall see Section 3.8.1, institutional actions have to be performed by means of suitable *declarations*.

We also introduce the fluent

$fluent\ Done(agent, action)$,

to represent that an action has been performed by an *agent*. In the paper we will sometime use the variable *actor:agent* to represent the agent that performs the action. For every type of action it is then necessary to introduce an axiom to initiate the relevant *Done* fluent when such an action takes place, for example:

Axiom Dn
$Initiates(Deliver(actor, agent, product), Done(actor, Deliver(actor, agent, product)), t)$

3.4 Temporal Propositions

Temporal propositions are propositions that become true or false over a predefined time interval, according to rules that will be specified below. The fluents used to represent temporal propositions exploit the following sorts (the axioms reported in brackets specify that the variables belonging to the corresponding sorts may assume a limited set of values):

$sort\ mode\ [\forall x : mode(x = Forall \vee x = Exist)]$
$sort\ ptime : integer\ [\forall x : ptime(x \geq 1)]$

We then introduce the sort *tp:fluent* of temporal propositions as a subsort of the sort *fluent*, and define the following function symbol used to represent temporal propositions:

$tp\ \ TP(fluent, ptime, ptime, mode)$

In the axioms we shall use variable *prop:fluent* (abbreviation of *proposition*) having sort *fluent* as the first argument of temporal propositions, and variables t_{start}:*ptime* and t_{end}:*ptime* of sort *ptime* to define a specific time interval. The fourth argument, of sort *mode*, is used to distinguish the case where it is required that the TP fluent holds at every instant between t_{start} and t_{end} (*Forall*) from the case where it is required that the TP fluent holds at least at one instant between t_{start} and t_{end} (*Exist*). For example, a temporal proposition with mode *Forall* can be used to state that the price of a certain product is x euro for the

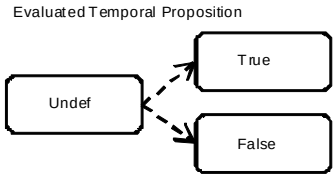

Figure 1: The life-cycle of evaluated temporal proposition.

current month, and a temporal proposition with mode *Exist* can be used to state that an agent will pay a certain amount of money to another agent in the current week.

Every time an agent uses a temporal proposition (for example in the exchange of a message) it has to initiate its fluent by means of the *AttCreateTP(fluent,ptime,ptime,mode)* (attempt to create a temporal proposition) action in order to be able to trace the evolution of the state of temporal propositions and therefore monitoring commitments fulfillment or violation. The *AttCreateTP()* action, if certain conditions are satisfied, initiates a new temporal proposition:

TP1 $(t_{start} \leq t_{end}) \wedge \neg HoldsAt(TP(prop, t_{start}, t_{end}, mode), t) \rightarrow$
$Initiates(AttCreateTP(prop, t_{start}, t_{end}, mode),$
$TP(prop, t_{start}, t_{end}, mode), t)$

3.5 Evaluated Temporal Proposition

Intuitively, at any given instant a temporal proposition may be $True$, $False$ or undefined ($Undef$) as depicted in Figure 1. For example, the temporal proposition it will rain today may be undefined at 4 pm; it may become true, say, at 6 pm, if it starts raining; or it may become false at midnight, if it has not rained for the whole day.

We introduce a new fluent, ETP, to represent an evaluated temporal proposition, that is, a temporal proposition with an attached truth value:

$sort\ value\ [\forall x : value(x = Undef \vee x = True \vee x = False)]$
$fluent\ \ ETP(tp, value)$

As depicted in Figure 1 a temporal proposition with mode $Exist$ usually (different cases are treated in the following) is initialized to $Undef$, it becomes $True$ if an event that initializes its *prop* happens between t_{start}-1 and t_{end}-1 (extremes included) (it means that *prop* starts to hold between t_{start} and t_{end}), otherwise at t_{end} it becomes $False$. Differently a temporal proposition with mode $Forall$ usually is initialized to $Undef$, it becomes $False$ if at t_{start} its *prop* does not hold or if an event that terminates its *prop* happens between t_{start} and t_{end}-1 (extremes included) otherwise at t_{end} it becomes $True$.

The action $AttCreateTP()$ that initiates a temporal proposition also initiates an evaluated temporal proposition ETP with the correct truth-value. In

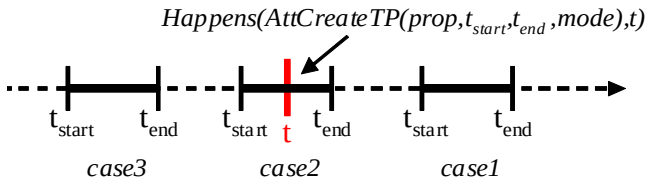

Figure 2: Temporal relation between t, the time when TP is created, and the time interval $[t_{start}, t_{end}]$

particular we have to distinguish the case where the temporal proposition is referred to the future (it is created at a time t that precedes t_{start}-1), from the case where it is referred to the past (it is created at a time t that follows t_{end}), from the case where it is created at a time t that is inside the interval of time defined by $t_{start} - 1$ and t_{end} as depicted in Figure 2. To make the axioms below more concise we will use the following definitions:

$e_{TP} =_{def} AttCreateTP(prop, t_{start}, t_{end}, mode)$
$condTP =_{def} (t_{start} \leq t_{end}) \wedge \neg HoldsAt(TP(prop, t_{start}, t_{end}, mode), t)$

Case 1. If a new temporal proposition is created at a time t that precedes t_{start}-1, indifferently from its *mode*, an *Undef* ETP has to be created:

ETP0 $t < (t_{start}\text{-}1) \wedge condTP \rightarrow$
$Initiates(e_{TP}, ETP(TP(prop, t_{start}, t_{end}, mode), Undef), t)$

As regard as the axioms for case 2 and case 3 and the axioms necessary to transform an $Undef$ evaluated temporal proposition in a $True$ or $False$ one, we have to clearly distinguish the temporal propositions with *mode* $Exist$ from the ones with *mode* $Forall$.

TP with mode $Exist$

Case 2. If a new temporal proposition is created at an instant t inside the interval determined by t_{start}-1 (included) and t_{end} an $Undef$ or a $True$ evaluated temporal proposition has to be initiated on the basis of what happened between t_{start}-1 and t_{end}:

ETPE2.1. $t_{start} - 1 \leq t < t_{end} \wedge condTP \wedge$
$\exists\, t_h, e_h(Happens(e_h, t_h) \wedge Initiates(e_h, prop, t_h) \wedge t_{start}\text{-}1 \leq t_h \leq t) \rightarrow$
$Initiates(e_{TP}, ETP(TP(prop, t_{start}, t_{end}, Exist), True), t)$

ETPE2.2. $t_{start} - 1 \leq t < t_{end} \wedge condTP \wedge$
$\neg\, \exists\, t_h, e_h(Happens(e_h, t_h) \wedge Initiates(e_h, prop, t_h) \wedge t_{start}\text{-}1 \leq t_h \leq t) \rightarrow$
$Initiates(e_{TP}, ETP(TP(prop, t_{start}, t_{end}, Exist), Undef), t)$

Case 3. If a new temporal proposition is created at a time t that is equal or follows t_{end}, a $True$ or a $False$ evaluated temporal proposition has to be created on the basis of what happened between t_{start}-1 and t_{end}-1 (included):

ETPE3.1. $condTP \wedge t \geq t_{end} \wedge$
$\exists t_h, e_h(Happens(e_h, t_h) \wedge Initiates(e_h, prop, t_h) \wedge t_{start}\text{-}1 \leq t_h < t_{end}) \rightarrow$
$Initiates(e_{TP}, ETP(TP(prop, t_{start}, t_{end}, Exist), True), t)$

ETPE3.2. $condTP \wedge t \geq t_{end} \wedge$
$\neg\exists t_h, e_h(Happens(e_h, t_h) \wedge Initiates(e_h, prop, t_h) \wedge t_{start}\text{-}1 \leq t_h < t_{end}) \rightarrow$
$Initiates(e_{TP}, ETP(TP(prop, t_{start}, t_{end}, Exist), False), t)$

The following axioms are used to change the truth-value of an existing $Undef$ evaluated temporal proposition with *mode Exist* on the basis of the temporal evolution of its *prop*. If an event e that initiates the *prop* happens within the interval of time defined by t_{start}-1 and t_{end}-1 (inclusive), the same event e also initiates an evaluated temporal proposition whose *value* is $True$:

ETPE1 $HoldsAt(ETP(TP(prop, t_{start}, t_{end}, Exist), Undef), t) \wedge$
$Initiates(e, prop, t) \wedge (t_{start} - 1 \leq t < t_{end}) \rightarrow$
$Initiates(e, ETP(TP(prop, t_{start}, t_{end}, Exist), True), t)$

If t_{end} is reached without the temporal proposition becoming *True*, passing time t_{end} initiates the evaluated temporal proposition with *value* equal to *False*. To represent the event that a certain instant of time is elapsed, we introduce a new event $Elapse(time)$ that we assume to happen every time a certain instant of time is reached as described by the following axiom:

A1 $Happens(Elapse(t), t)$

ETPE2 $HoldsAt(ETP(TP(prop, t_{start}, t_{end}, Exist), Undef), t) \rightarrow$
$Initiates(Elapse(t_{end}), ETP(TP(prop, t_{start}, t_{end}, Exist), False), t)$

Axioms of the form of ETPE1, where the *Initiates()* predicate is in the antecedent of an axiom, are useful to resolve the *ramification problem*, that is the situations where one event has some indirect effects, for example in this case event e has the effect to initiate *prop* and also the effect to transform the related *ETP* from *Undef* to $True$. As discussed in [30, pag 110] unfortunately it is impossible to compute circumscription of this type of axioms. Therefore in the actual specification of a system, if we have a finite set of axioms that state that E_1,...,E_n initiate $Prop1$, we will have to transform ETPE1 in n different axioms obtained by removing *Initiates()* from the antecedent and using E_1, or E_2, or E_n in the *Initiates()* of the consequent. This operation has to be done for every axioms that has $Initiates(e, p, t)$ in the antecedent.

In a similar way it is possible to write the axioms for the temporal evolution of evaluated temporal propositions with mode $Forall$.

Every time that a $True$ or $False$ evaluated temporal proposition is initiated by the event e, the corresponding *Undef* proposition is terminated by the same event, indifferently from its mode:

ETP1
$Initiates(e, ETP(tp, True), t) \rightarrow Terminates(e, ETP(tp, Undef), t)$

ETP2
$Initiates(e, ETP(tp, False), t) \rightarrow Terminates(e, ETP(tp, Undef), t)$

Similarly to axiom ETPE1, to make it possible to compute circumscription, $ETP1$ and $ETP2$, having $Initiates$ in the antecedent, should be transformed in axioms where event e in the $Terminates$ predicate is replaced by all possible concrete events that in the actual specification of the system make the antecedent true.

Finally, we introduce with the following axiom the identically *True* evaluated temporal proposition, that initially holds and is never terminated. Its proposition is represented by introducing the constant $PTrue$ that has sort *fluent*.

$$HoldsAt(ETP(TP(PTrue, 1, 1, Exist), True), 0)$$

3.6 Commitment and Precommitment

Commitment, precommitment, and conditional commitment will be used in Section 3.7 to express the semantics of a library of communicative acts and will be used in Section 3.11 to express active norms and to monitor the fulfillment or violation of obligations and prohibitions of the various agents at run-time.

3.6.1 Commitment

The fluent used to represent commitments, whose life-cycle is depicted in Figure 3, is as follows:

$fluent\ Comm(state, agent, agent, tp, tp, source, id)$.

In the axioms we shall use the more perspicuous variables name *debtor*:*agent* and *creditor*:*agent* as the second and third argument, and *content*:*tp* and *condition*:*tp* as the fourth and fifth argument. The sort used to represent the *state* of a commitment is:

$$sort\ state\ [\forall x : state\ (x = Active \vee x = Cond \vee x = Pending \vee x = Cancelled \vee x = Fulfilled \vee x = Violated)].$$

The sort used to represent the *source* of a commitment used to distinguish a commitment created by a communicative act from a commitment created by the activation or violation of a norm is:

$sort\ source\ [\forall x : source\ (x = CA \vee x = Norm \vee x = Sanction)]$.

Finally the sort used to represent the *id* of the norm that created the commitment or the time instant when the communicative act that generated the commitment has been performed:

$sort\ id : integer\ [\forall x : id(x \geq 1)]$

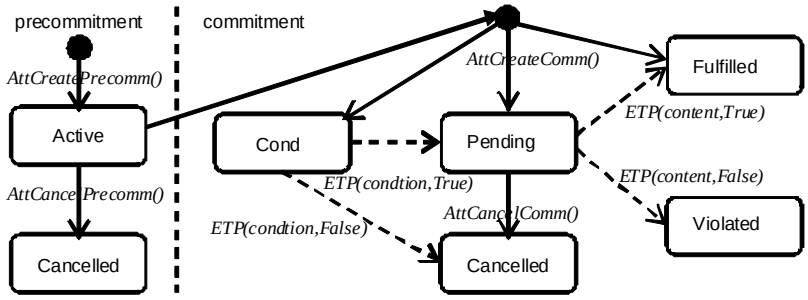

Figure 3: The life-cycle of precommitment and commitment.

Differently from other approaches [41], we introduce one fluent to represent both conditional and unconditional commitment, so that we do not need to define different axioms and different communicative acts to create two types of commitment and to update their states. Another advantage is that information about the condition can be kept during the life-cycle of the commitment: in fact, all information about the condition is lost if the conditional commitment is transformed into an unconditional commitment (like in [41]) when the condition starts to hold.

Before creating a commitment a certain set of conditions have to be verified (for example the time interval of the *condition* must precede the time interval of the *content*) and on the basis of the truth-value of the temporal propositions used as *content* and as *condition* a different commitment has to be created. Therefore, to improve the modularity of the system we introduce the institutional action *AttCreateComm(agent,agent,tp,tp,source,id)* (attempt to create a commitment), whose effect, when successful, is to create a commitment as stated in the axioms reported below.

As regard as the *content* and the *condition* of a commitment, it is possible to have different situations when a new commitment is created: the truth-value of the *condition* and of the *content* are $Undef$, or the truth-value of the *condition* is $True$ (or initiated to $True$ contemporary to the commitment creation) and the truth-value of the *content* is $Undef$ or $True$ (or initiated to $True$ contemporary to the commitment creation). Abbreviations: *d=debtor, c=creditor, s=source*,
$content \equiv_{def} TP(prop1, t1_{start}, t1_{end}, mode1)$,
$condition \equiv_{def} TP(prop2, t2_{start}, t2_{end}, mode2)$.

C01

$t2_{end} < t1_{start} \wedge \neg HoldsAt(Comm(state, d, c, content, tp2, s, id), t) \wedge$
$HoldsAt(ETP(content, Undef), t) \wedge HoldsAt(ETP(condition, Undef), t) \wedge$
$\neg\exists e(Happens(e, t) \wedge Initiates(e, ETP(condition, True), t)) \rightarrow$
$Initiates(AttCreateComm(d, c, content, condition, s, id),$
$Comm(Cond, d, c, content, condition, s, id), t)$

.

C02
$t2_{end} < t1_{start} \wedge \neg HoldsAt(Comm(state, d, c, content, condition, s, id), t) \wedge$
$HoldsAt(ETP(content, Undef), t) \wedge HoldsAt(ETP(condition, Undef), t) \wedge$
$\exists e(Happens(e, t) \wedge Initiates(e, ETP(condition, True), t)) \rightarrow$
$Initiates(AttCreateComm(d, c, content, condition, s, id),$
$Comm(Pending, d, c, content, condition, s, id), t)$

If the *condition* is already $True$ it is necessary to distinguish different cases on the basis of the truth-value of the *content*:

C03
$t2_{end} < t1_{start} \wedge \neg HoldsAt(Comm(state, d, c, content, condition, s, id), t) \wedge$
$HoldsAt(ETP(content, Undef), t) \wedge HoldsAt(ETP(condition, True), t) \wedge$
$\neg\exists e(Happens(e, t) \wedge Initiates(e, ETP(content, True), t)) \rightarrow$
$Initiates(AttCreateComm(d, c, content, condition, s, id),$
$Comm(Pending, d, c, content, condition, s, id), t)$

C04
$t2_{end} < t1_{start} \wedge \neg HoldsAt(Comm(state, d, c, content, condition, s, id), t) \wedge$
$HoldsAt(ETP(content, Undef), t) \wedge HoldsAt(ETP(condition, True), t) \wedge$
$\exists e(Happens(e, t) \wedge Initiates(e, ETP(content, True), t)) \rightarrow$
$Initiates(AttCreateComm(d, c, content, condition, s, id),$
$Comm(Fulfilled, d, c, content, conditions, id), t)$

A *Fulfilled* commitment has to be created if the *content* and the *condition* are already $True$, for example in the case of commitments that derive from assertions, like "yesterday it rained".

C05
$t2_{end} < t1_{start} \wedge \neg HoldsAt(Comm(state, d, c, content, condition, s, id), t) \wedge$
$HoldsAt(ETP(content, True), t) \wedge HoldsAt(ETP(condition, True), t) \rightarrow$
$Initiates(AttCreateComm(d, c, content, condition, s, id),$
$Comm(Fulfilled, d, c, content, condition, s, id), t)$

In the life-cycle of a commitment (Figure 3), dotted lines represent state changes due to events happening in the system that modify the truth-value of the *content* or of the *condition* of the commitment, as described in the following axioms. A $Cond$ commitment becomes $Pending$ when the evaluated temporal proposition of its *condition* with *value True* starts to hold; if the evaluated temporal proposition of the *condition* with *value False* starts to hold, the $Cond$ commitment becomes *Cancelled*:

C1 $HoldsAt(Comm(Cond, d, c, content, condition, s, id), t) \wedge$
$Initiates(e, ETP(condition, True), t) \rightarrow$
$Initiates(e, Comm(Pending, d, c, content, condition, s, id), t)$

C2 $HoldsAt(Comm(Cond, d, c, content, condition, s, id), t) \wedge$
$Initiates(e, ETP(condition, False), t) \rightarrow$
$Initiates(e, Comm(Cancelled, d, c, content, condition, s, id), t)$

A *Pending* commitment becomes *Fulfilled* when the evaluated temporal proposition of its *content* with truth-value *True* starts to hold:

C3 $HoldsAt(Comm(Pending, d, c, content, condition, s, id), t) \wedge$
$Initiates(e, ETP(content, True), t) \rightarrow$
$Initiates(e, Comm(Fulfilled, d, c, content, condition, s, id), t)$

A *Pending* commitment becomes *Violated* when the evaluated temporal proposition of its *content* with truth-value *False* starts to hold:

C4 $HoldsAt(Comm(Pending, d, c, content, condition, s, id), t) \wedge$
$Initiates(e, ETP(content, False), t) \rightarrow$
$Initiates(e, Comm(Violated, d, c, content, condition, s, id), t)$

To terminate a commitment whose state has been replaced by another one, we introduce a predicate *Replace(state1,state2)*, that is true if *state1* can be reached from *state2*. In particular the following predicates are true:

$Replace(Pending, Cond)$ $Replace(Cancelled, Cond)$
$Replace(Cancelled, Pending)$ $Replace(Fulfilled, Pending)$
$Replace(Violated, Pending)$

We then define:

C5 $Initiates(e, Comm(state1, d, c, content, condition, s, id), t) \wedge$
$HoldsAt(Comm(state2, d, c, content, condition, s, id), t) \wedge$
$Replace(state1, state2) \rightarrow$
$Terminates(e, Comm(state2, d, c, content, condition, s, id), t)$

Another institutional action that manipulates commitments is *AttCancelComm(agent,agent,tp,tp)* (attempt to cancel a commitment), that transforms a *Cond* or *Pending* commitment into a *Cancelled* one and its effect is defined by the following axiom:

C6 $\exists\, state(state = Cond \vee state = Pending) \wedge HoldsAt(Comm(state,$
$d, c, content, condition, s, id), t) \rightarrow Initiates(AttCancelComm(d, c$
$content, condition), Comm(Cancelled, d, c, content, condition, s, id), t).$

3.6.2 Precommitment

The commitment defined so far can be used to express the meaning of various speech acts like *assertions* and *promises*. However, it is not possible to express the meaning of directive speech acts, like *requests*. When an agent requests another agent to do something, it is trying to induce the other agent to make a commitment, and the other agent can commit itself by just *accepting* the request. We represent this situation by means of *precommitment*, a type of commitment that may also play a crucial role in the phase of negotiation of a new commitment when more than two agents are involved. A precommitment is similar to a commitment, but has one more attribute: the time that may elapse

between the creation of the precommitment and action of accepting or refusing it, which is represented in the axioms by means of the variable t_{out}:*ptime*. The following fluent, whose life-cycle is depicted in Figure 3, is used to represent precommitments:

$fluent\ Precomm(state, agent, agent, tp, tp, ptime, source, id)$

To create a precommitment the *AttCreatePrecomm(agent,agent,tp,tp,ptime,source,id)* institutional action has to be performed. Its effects are described by the following axiom (where *content* and *condition* are defined in the previous section):

P1 $\neg HoldsAt(Precomm(state, d, c, content, condition, t_{out}, s, id), t) \wedge t < t_{out} \wedge$
$t2_{end} < t1_{start} \rightarrow Initiates(AttCreatePrecomm(d, c, content, condition, t_{out}, s, id),$
$Precomm(Active, d, c, content, condition, t_{out}, s, id), t)$

The action *AttAcceptPrecomm(agent,agent,tp,tp)* of accepting an existing *Active* precommitment implies the creation of a new commitment:

P2 $Happens(AttAcceptPrecomm(d, c, content, condition), t) \wedge$
$\exists\, t_{out}\ (HoldsAt(Precomm(Active, d, c, content, condition, t_{out}, s, id), t) \wedge t < t_{out}) \rightarrow$
$Happens(AttCreateComm(d, c, content, condition, s, id), t)$

If a precommitment is accepted it is terminated:

P3 $HoldsAt(Precomm(Active, d, c, content, condition, t_{out}, s, id), t) \wedge t < t_{out} \rightarrow$
$Terminates(AttAcceptPrecomm(d, c, content, condition),$
$Precomm(Active, d, c, content, condition, t_{out}, s, id), t)$

We need moreover to define the *AttCancelPrecomm(agent,agent,tp,tp)* action that the *debtor* of a precommitment can perform to refuse a precommitment, transforming it into a *Cancelled* one and terminating the *Active* one:

P4 $HoldsAt(Precomm(Active, d, c, content, condition, t_{out}, s, id), t) \wedge t < t_{out} \rightarrow$
$Initiates(AttCancelPrecomm(d, c, content, condition),$
$Precomm(Cancelled, d, c, content, condition, t_{out}, s, id), t)$

P5 $HoldsAt(Precomm(Active, d, c, content, condition, t_{out}, s, id), t) \wedge t < t_{out} \rightarrow$
$Terminates(AttCancelPrecomm(d, c, content, condition),$
$Precomm(Active, d, c, content, condition, t_{out}, s, id), t)$

Similarly if t_{out} is elapsed without the precommitment being accepted (i.e., it is still *Active*) it has to be transformed into a *Cancelled* one and the *Active* one has to be terminated:

P6 $HoldsAt(Precomm(Active, d, c, content, condition, t_{out}, s, id), t) \rightarrow$
$Initiates(Elapse(t_{out}), Precomm(Cancelled, d, c, content, condition, t_{out}, s, id), t)$

P7 $HoldsAt(Precomm(Active, d, c, content, condition, t_{out}, s, id), t) \rightarrow$
$Terminates(Elapse(t_{out}), Precomm(Active, d, c, content, condition, t_{out}, s, id), t)$

3.7 Agent Communication Language

In this section we define a library of agent speech acts. Following Bach and Harnish [4], we distinguish between two types of speech acts: *communicative acts*, similar to those defined by FIPA ACL communicative acts library [15], and *declarative acts* or *declarations*. Speech acts of the first type (like promising, informing, and requesting), are important because reflect the everyday use of language. Declarative communicative acts, usually neglected by other approaches [15, 14, 37], are also important because by means of them it is possible to perform institutional actions, like opening an auction or giving the power to perform certain actions to an agent. We define the following sort that is used to specify the type of the message exchanged between two agents:

$$sort\ type\ [\forall x : type\ (x = Promise \vee x = Inform \vee x = Request \vee x = Agree \vee x = Refuse \vee x = Cancel \vee x = Declare)]$$

3.7.1 Communicative acts

In our model we define the following base-level actions to represent the exchange of a certain message, where the second type of message exchange is necessary for the performance of requests with a specified deadline for acceptance or refusal:
$ExchMsg(type, agent, agent, tp, tp)$
$ExchMsg1(type, agent, agent, tp, tp, ptime)$

In the axioms we will use the more perspicuous variables name *sender*:*agent* and *receiver*:*agent* as second and third argument of a message. The performance of a *Promise* communicative act implies the attempt to create a commitment, provided that the *content* is referred to the future (i.e., $t1_{start}$ is greater than the time t of the performance of the act), and represents the performance of an action by the *sender* of the message (*content* and *condition* are the two temporal propositions defined in Section 3.6):

Axiom Promise
$Happens(ExchMsg(Promise, sender, receiver, content, condition), t) \wedge$
$(t1_{start} > t) \wedge prop1 = Done(sender, action) \rightarrow$
$Happens(AttCreateComm(sender, receiver, content, condition, CA, t), t)$

The performance of an *Inform* communicative act implies the attempt to create a commitment:

Axiom Inform
$Happens(ExchMsg(Inform, sender, receiver, content, condition), t) \rightarrow$
$Happens(AttCreateComm(sender, receiver, content, condition, CA, t), t)$

It is worth to observe that an act of informing can be about something happened in the past or something that will happen in the future. An *Inform* act about the performance of a future action will have the same consequences as a promise. To distinguish between promising to perform an action and informing

that an action will be performed, it would be necessary to take into consideration aspects, like the agents interests, that concern the internal architecture of the interacting agents; this is beyond the scope of this chapter and would reduce the possibility of applying our model to arbitrary open systems.

The performance of a *Request* communicative act implies the attempt to create a precommitment (with the *receiver* of the message as *debtor* of the upcoming commitment), provided that the *content* is referred to the future and is about the performance of an action by the *receiver* of the message:

Axiom Request
$Happens(ExchMsg1(Request, sender, receiver, content, condition, t_{out}), t) \wedge$
$(t1_{start} > t) \wedge prop1 = Done(receiver, action) \rightarrow$
$Happens(AttCreatePrecomm(receiver, sender, content, condition, t_{out}, CA, t), t)$

The meaning of an *Agree* communicative act is expressed by means of the *AttAcceptPrecomm()* action that, if the accepted precommitment exists, transforms it into a commitment:

Axiom Agree
$Happens(ExchMsg(Agree, sender, receiver, content, condition), t) \rightarrow$
$Happens(AttAcceptPrecomm(sender, receiver, content, condition), t)$

The meaning of a *Refuse* communicative act is expressed by means of the *AttCancelPrecomm()* action that transforms the precommitment, if it exists, into a *Cancelled* one:

Axiom Refuse
$Happens(ExchMsg(Refuse, sender, receiver, content, condition), t) \rightarrow$
$Happens(AttCancelPrecomm(sender, receiver, content, condition), t)$

The meaning of a *Cancel* communicative act is expressed by means of the *AttCancelComm()* action that transforms a *Cond* or *Pending* commitment, having as *creditor* the *sender* of the message, into a *Cancelled* one:

Axiom Cancel
$Happens(ExchMsg(Cancel, sender, receiver, content, condition), t) \rightarrow$
$Happens(AttCancelComm(receiver, sender, content, condition), t)$

3.8 Declarations

Declarations are a very special type of speech act that, if certain contextual conditions hold, change the institutional state of the system, that is, the part of the system that exists only thanks to the common agreement of the interacting agents (or of their designers). We view an artificial interaction system as a technological instrument that enriches and creates new types of interaction among human beings. Such a system has to represent: (i) *physical objects* that exist in human reality, like a book that has to be sold in an auction; (ii) *institutional entities* of human reality, like the amount of money that has to be paid to buy

a book; (iii) *institutional entities* created and managed within the artificial system to implement communicative interactions among agents, like for instance the concept of commitment, the notion of auction, or the notion of price. These institutional entities may be modified by declarations. For instance, the declaration that an electronic auction is open, made by an agent that has the power to do it, makes it actually open.

To deal with institutional entities we introduce *institutional actions*, which are a special type of actions [10] and are crucial for the formalization of declarations. Given that the performance of institutional actions have to be *public* (that, is made known to the relevant agents), we assume that to perform an institutional action an agent has to perform a declaration by sending a particular type of message. Following Searle's approach to the construction of social reality [33], we assume that the *"counts-as"* relation binds the performance of a base-level action, like sending a message, to the performance of an institutional action, if certain contextual conditions are satisfied. We deal with such contextual conditions in the next subsection.

3.8.1 Contextual conditions and Power

The most important condition that has to be satisfied for a declaration to count as the declared institutional action is that the *sender* of the message has the *power* to perform the institutional action. We therefore provide a specific treatment for this aspect of the context. Assuming that suitable institutional actions have been defined (with sort *iaction*, a subsort of *event*), to express the fact that an *agent* has the power to perform an institutional action we introduce the fluent:

$$fluent\ Power(agent, iaction)$$

For example, we may want to specify that agent *Bob* has the power to perform the institutional action $AttCreateComm()$ with itself as the *creditor*, agent *Mark* as the *debtor* (*Bob* may be *Mark*'s boss), and as *content* a temporal proposition representing that some photocopies are made from instant 2 to instant 5 (without any further *condition*):

$$HoldsAt(Power(Bob, AttCreateComm(Mark, Bob, TP(Done(Mark, Photocopy), 2, 5, Exist), TP(PTrue, 1, 1, Exist))), 0)$$

To specify all other relevant aspects of the context of a declaration we introduce another fluent, *Context(iaction)*, that holds if certain contextual conditions are satisfied. Such a fluent may initially hold or may be initiated by a suitable axiom. For example, we may want to represent that an auction (represented with the fluent *Auction(state,id,* t_{init}*)*) may be opened by the institutional action *OpenAuction(id)* only if a certain instant of time t_{init} has elapsed:

$$Initiates(Elapse(t_{init}), Context(OpenAuction(id)), t)$$

Before introducing an axiom to define the effects of a declaration we first introduce a new base-level action for exchanging a message of type declare:

$ExchMsgD(Declare, agent, agent, iaction)$

The communicative act of declaring a certain institutional action counts as the performance of the declared institutional action if certain conditions hold, and if the *sender* of the message has the power to perform the declared institutional action, as described by the following axiom:

Axiom Decl
$Happens(ExchMsgD(Declare, sender, agent, iaction), t) \wedge$
$HoldsAt(Power(sender, iaction), t) \wedge HoldsAt(Context(iaction), t) \rightarrow$
$Happens(iaction, t)$

3.8.2 Empower and Disempower

Given that the power is a fluent, it would be natural to introduce a new institutional action that can be used during the run time of a system to *empower* and another to *disempower* a given agent to perform a given institutional action. They are institutional actions because they change an institutional fluent, the power, that exists thanks to the common agreement of the interacting agents. Moreover given that they are institutional actions, their effects will take place only if the actor has the power to perform them. The *Empower()* and *Disempower()* institutional actions are defined as follows (where the first agent (for which we will use the variable *actor:agent*, is the one that gives/removes the power to/from the second agent):

$iaction\ Empower(gent, agent, iaction)$
$iaction\ Disempower(agent, agent, iaction)$

The effect of performing an *Empower()* institutional action is given by the following axiom:

Axiom Power1
$Initiates(Empower(actor, agent, iaction), Power(agent, iaction), t)$

The effect of performing a *Disempower()* action is to terminate a given power:

Axiom Power2
$HoldsAt(Power(agent, iaction), t) \rightarrow$
$Terminates(Disempower(actor, agent, iaction), Power(agent, iaction), t)$

To avoid infinite chains of empowerments, it is necessary to state what powers initially hold when the system start to run, that is, what powers hold at time 0. For example, agent *Bob* may initially have the power to perform an $AttCreateComm()$ action with every possible agent as *creditor* and every possible *content* and *condition* as described by the following axiom:

$HoldsAt(Power(Bob, AttCreateComm(Bob, creditor, content, condition)), 0)$

This approach has the limitation that usually it is impossible to know at design time the name of the actual agents. As we will see in next section, this problem may be overcome by introducing the notion of *role*. The same power can also be created at time t by an agent $Ag1$ (if it has the right power) by declaring the following action:

$$Empower(Ag1, Bob, AttCreateComm(Bob, creditor, content, condition))$$

3.9 Role

The set of powers and norms (see Section 3.11) that hold in an interaction system at a certain instant of time may be created (like the other fluents introduced so far) by the interacting agents using the proper institutional action (actually in this work we will not define the institutional action for creating norms at run-time) or may be devised by the (human) designers of the system at design time. Given that at design time it is impossible to know the concrete agents that will be actually involved at run time in an interaction we need to introduce the notion of *role* and to introduce some mechanisms to define powers and norms in term of roles. Moreover we need to make it possible during the run time phase to deduce powers and norms that apply to a given agents on the basis of the roles that it plays.

As regard as the notion of role, in an artificial system we could have for example the role of *auctioneer* or *participant* of a certain run of an auction, *boss* or *employee* of a certain company for example an Auction House, *debtor* or *creditor* of a certain commitment, *student* or *professor* of a certain university etc. Coherently with those examples in our view a role may be viewed as a label that can be used in place of the identifier of a specific agent in the design of a certain institution, and it may be related to: a specific institution/organization or institutional agent (like a university), to an institutional activity (like a run of an auction), or to an institutional relation (like a commitment), that is, a role relates an agent with an institutional "*entity*". In literature it is possible to find different other definitions of the notion of role, for an overview of those approaches see [28].

It is important to notice that during run-time some roles must necessarily be played by certain agents. For example it makes no sense having a commitment without an agent playing the role of *debtor* and another one playing the role of *creditor*; similarly it makes no sense having a run of an auction without an agent playing the role of *auctioneer* and at least other two other agents as *participants*.

We assume that every institutional entity is explicitly represented in our model, and we introduce the sort *ientity:fluent* as a subsort of the sort *fluent* to represents them. For example we have already introduced the fluent to represent commitments, we may define the following fluent to represent a run of an auction (where the sort *astate* (auction state) may assume the values: *Registration, Open, Closed*, and we will use the variable t_{init}*:ptime* as third argument):

$$ientity\ Auction(id, astate, ptime)$$

or a fluent to represent an organization:

$ientity\ Organization(id)$

To express the fact that an agent plays a given role within a certain institutional entity, we introduce the following fluent:

$fluent\ HasRole(agent, role, ientity)$

where the *sort role* is used to represent names of possible roles like *Debtor*, *Creditor*, etc. For example if the following fluent holds at time t:

$HoldsAt(HasRole(Bob, Employer, Organization(01)), t)$

it means that agent *Bob* play the role of *Employer* in the organization with identifier 01 a time t. Moreover, to express that a certain role is meaningful only in relation to a certain institutional entity we introduce the following fluent:

$fluent\ RoleOf(role, ientity)$

For example, to state that the role *Auctioneer* is meaningful in the context of the run of an auction the following fluent has to be initiated:

$RoleOf(Auctioneer, Auction(id, astate, t_{init}))$

It is important to remark that every time that a new institutional entity is initiated it is also necessary to initiate the set of roles that it defines (that is a set of *RoleOf()* fluents) and, if it is the case, the correct *HasRole()* fluents. For example when a new commitment fluent is initiated, it is also necessary to initiates the $RoleOf()$ fluent that defines the role of *Debtor* and *Creditor* and to initiate new *HasRole()* fluents to state that the first agent that appear in the new commitment plays the role of *Debtor* and the second one plays the role of *Creditor*. Therefore the following axioms have to be introduced (where $e_c =_{def} AttCreateComm(agent1, agent2, content, condition, s, id)$:

$Initiates(e_c, RoleOf(Creditor, Comm(agent1, agent2, content, condition, s, id)), t)$

$Initiates(e_c, RoleOf(Debtor, Comm(agent1, agent2, content, condition, s, id)), t)$

$Initiates(e_c, HasRole(agent1, Debtor, Comm(agent1, agent2, content, condition, s, id)), t)$

$Initiates(e_c, HasRole(agent2, Creditor, Comm(agent1, agent2, content, condition, s, id)), t)$

In other situations, like for example for the roles defined by a run of an *Auction*, it is not be necessary to assign all the roles to specific agents when the institutional entity is created, but it has to be possible to assign the role of *Auctioneer* and *Participant* to various agents subsequently. In order to be able to perform such an institutional action we introduce the *AssignRole(agent,agent,role,ientity)* action that when performed assign the new role to the involved agent, as described by the following axiom:

Axiom Role1. $HoldsAt(RoleOf(role, ientity), t) \rightarrow$
$Initiates(AssignRole(actor, agent, role, ientity), HasRole(agent, role, ientity), t)$

and another action *DismissRole(agent,agent,role,ientity)* to remove an agent from a given role:

Axiom Role2. $HoldsAt(HasRole(agent, role, ientity), t) \rightarrow$
$Terminates(DismissRole(actor, agent, role, ientity), HasRole(agent, role, ientity), t)$

Given that $AssignRole()$ and $DismissRole()$ are two institutional actions, their *actor* needs to have the right powers to successfully perform them. As performing such actions through declarations is the only way to assign or dismiss roles, we have the problem of initially assigning a role to each agent. To solve this problem we introduce a special agent, the *interaction-system* (*IntSystem*), and assume that it has the power to assign every role and every power to other agents and that the *Context()* to perform those actions initially holds.

3.10 Conditional Power

In Section 3.8.1 we introduced the notion of power; here we introduce the general notion of conditional power and a crucial type of conditional power, conditioned by the roles that agents play in the system. In the design of an artificial institution it may be useful to specify conditional powers, that is, powers that starts to hold if certain conditions start to hold or if certain events happen in the system. For example agent *Bob* may acquire the power to create commitments for himself when he becomes of age. To initiate a power when a given event E_i happens in the system, we have to add an axiom like the following one to our specification:

$$Happens(E_i, t) \rightarrow Happens(Empower(IntSystem, Agent, Iaction), t)$$

Obviously such power may be terminated by another event E_t as stated by the following axiom:

$$Happens(E_t, t) \rightarrow Happens(Disempower(IntSystem, Agent, Iaction), t)$$

In general if n events, with $n > 1$, must happen for a power to be created, and we do not know their temporal order, we need to write n axioms to initiate the right power. For example the following two axioms are needed when two events must happen:

$$Happens(E_1, t_1) \wedge\ t_1 <= t \wedge Happens(E_2, t) \rightarrow \\ Happens(Empower(IntSystem, Agent, Iaction), t)$$

$$Happens(E_2, t_1) \wedge\ t_1 <= t \wedge Happens(E_1, t) \rightarrow \\ Happens(Empower(IntSystem, Agent, Iaction), t)$$

3.10.1 Power expressed by means of roles

An important type of conditional power is the power conditioned by the fact that the involved agents play certain roles. In the design phase of a system, to express that a power is initiated when certain agents start to play some role it is necessary to introduce axioms similar to the ones presented in the previous section. For example the following axiom expresses the fact that the agent playing the role of *Boss* of an organization has the power to create commitments for the agents playing the role of *Employee* in the same organization (in this axiom, an agent becomes the *Boss* first, and then another agent becomes the *Employee*; an axiom treating the opposite order of events has also to be introduced):

$$Happens(AssignRole(IntSystem, agent1, Boss, Organization(1)), t_1) \wedge t_1 <= t \wedge$$
$$Happens(AssignRole(IntSystem, agent2, Employee, Organization(1)), t) \rightarrow$$
$$Happens(Empower(IntSystem, agent1, AttCreateComm(agent2, agent1, content, condition)), t)$$

Similarly it is necessary to add the following axiom to dismiss the power when an agent dismisses the relevant role:

$$HoldsAt(Power(agent1, AttCreateComm(agent2, agent1, content, condition)), t) \wedge$$
$$(Happens(DismissRole(IntSystem, agent1, Boss, Organization(1)), t) \vee$$
$$Happens(DismissRole(IntSystem, agent2, Employee, Organization(1)), t) \rightarrow$$
$$Happens(DisempowerRole(agent1, AttCreateComm(agent2, agent1, content, condition)), t)$$

Commitment and Precommitment Powers

Institutional actions like *AttCreateComm()*, *AttCancelComm()* can be performed by exchanging a message of a particular type (for example a promise) or can be declared. In the latter case, for the declaration to be successful the sender of the message must have the power to declare those actions. Therefore we have to introduce some axioms to initiate reasonable powers. First of all, every agent has the power to perform an $AttCreateComm()$ institutional action with itself as the debtor (first argument) and any other agent as the creditor (second argument) of the future commitment:

Axiom CommPower1 $agent1 \neq agent2 \rightarrow$
$$Happens(Empower(IntSystem, agent1, AttCreateComm(agent1, agent2, content, condition, s, id)), 0)$$

The agent that becomes the creditor of a commitment will have the power to cancel it:

Axiom CommPower2
$$Happens(AttCreateComm(agent1, agent2, content, condition, s, id), t) \rightarrow$$
$$Happens(Empower(InsSystem, agent2, AttCancelComm(agent1, agent2, content, condition)), t)$$

Moreover, the agent that becomes the debtor of a precommitment will have the power to accept or refuse it:

Axiom PrecommPower1
$$Happens(AttCreatePrecomm(agent1, agent2, content, condition, t_{out}), t) \rightarrow$$
$$Happens(Empower(IntSystem, agent1, AttAcceptPrecomm(agent1, agent2, content, condition), t)$$

Axiom PrecommPower2
$Happens(AttCreatePrecomm(agent1, agent2, content, condition, t_{out}), t) \rightarrow$
$Happens(Empower(IntSystem, agent1, AttCancelPrecomm(agent1, agent2, content, condition), t)$

3.11 Norms

A norm is used to impose a certain behavior on certain agents in the system identified by means of the norm's *debtor*. The norm *content* is a temporal proposition (see Section 3.4) that describes the actions that the debtor have to perform (if the norm expresses an *obligation*) or not to perform (if the norm expresses a *prohibition*) within a specified interval of time. In our model if for a given agent an action is not obliged nor prohibited it is permitted, obviously if it is an institutional action the agent need also to have the power to perform it in order that its effects take place (in particular the value of the attribute t_{start} of the *content* is always equal to the time of occurrence of the event that activates the norm). The obligation or prohibition could be conditioned to the truth of another temporal proposition indicated in the *condition* attribute of the norm.

An agent can reason whether to fulfill or not to fulfill a norm on the basis of the sanctions (as discussed later) and of whom is the *creditor* of the norm, as proposed also in [25, 27]. For example, an agent with the role of auctioneer may decide to violate a norm imposed by the auction house if it is in conflict with another norm that regulates trade transactions in a certain country. Moreover the creditor of a norm is crucial because, given that it becomes the creditor of the commitments generated by the norm, it is the only agent authorized to cancel such commitment.

To enforce norms it is necessary to specify *sanctions*. Regarding this aspect there are numerous reasons to develop systems where agents may violate the rules, first of all as discussed in [17], the obligation to perform an action in principle cannot be regimented[2], secondly systems that are able to manage violations (sometimes also due to software error) are more robust, finally it is important to remark that given that it is impossible to predict at design phase all the interesting and fruitful behaviors that may emerge in an interaction, to reach an optimal solution for all participants [42], it may be profitable to allow agents to violate their obligations and prohibitions. We speak about sanctions to stress the need to punish violations, but the mechanisms for the management of *rewards* can be easily introduced in a similar way. As discussed in [17] and differently from other approaches [27], [39], [22] that do not investigate in detail this aspect, we think that a complete model of sanctions has to distinguish between two different type of actions: the action that the violator of a norm has to perform to extinguish its violation and the action that the agent in charge of norm enforcement may perform to deter agents from violating the norm. Therefore a norm has to specify: (i) what we call the *active sanction* (*a-sanction*) that

[2]With regimentation [23] we mean the introduction of control mechanisms that does not allow agents to violate obligations or prohibitions.

is the action that the violator should perform within a certain interval of time to extinguish its violation and that can be represented in our model through a temporal proposition; (ii) what we call the *passive sanction* (*p-sanction*) that is the new power that the agent entitled to enforce the norm acquire in case of violation of the norm. Regarding this second type of action, it is important to underline that in case of violation the agent entitled to enforce the norm gets a new power whereas another norm (that in [27] is called *enforcement norm*) may oblige the enforcer agent to punish the violation.

A norm becomes active every time that its *activation event* e_{start} happens. For example a norm may be used to represent a contract that obliges an organization to pay the salary to its employees every time that the end of the month is reached, or a set of norms may be used to formalize a protocol, for example an auction protocol, and oblige the auctioneer to declare the new ask price every time that a new valid bid is received, or to declare the winner when a certain amount of time is elapsed without new bids. The activation of a norm arises the need to monitor its evolution in time and to react to its *violation* with suitable *sanctions*. Given that in our meta-model we have already defined a construct, the social commitment, which can be used to perform that task, we define a set of axioms that transform a norm into a commitment when its activation event happens. This makes it possible to resolve another interesting problem: the problem to detect and manage the violation or fulfillment of more than one activation of the same norm. We introduce the following fluent to represent norms:

$fluent\ Norm(id, agent, agent, tp, tp, event, tp, fluent)$

In the axioms on norms we shall use variables *debtor:agent* and *creditor:agent* as the second and third argument of a norm, variables *content:tp* and *condition:tp* as the fourth and fifth argument, variable e_{start}*:event* to refer to the activation event, the variable *a-sanction:tp* for the active sanction, and variable *p-sanction:fluent* for the passibe sanction.

We need therefore to define an axiom to manage the creation of a commitment every time that a norm becomes active (abbreviations: *d=debtor, c=creditor*):

Axiom ActivateNorm
$HoldsAt(Norm(id, d, c, content, condition, e_{start}, a\text{-}sanction, p\text{-}sanction), t) \wedge$
$Happens(e_{start}, t) \rightarrow$
$Happens(AttCreateComm(d, c, content, condition, Norm, id), t)$

If event e initiates the commitment related to a norm to *Violated*, that event e initiates also the power, described in the *passive-sanction*, for certain agent to perform the actions that have been devised to deter the agent from misbehaving, as described by the following axiom:

.

Axiom PassiveSanction
$HoldsAt(Norm(id, d, c, content, condition, e_{start}, a\text{-}sanction, p\text{-}sanction), t) \wedge$
$Initiates(e, Comm(Violated, d, c, content, condition, Norm, id), t) \rightarrow$
$Initiates(e, p\text{-}sanction, t)$

If event e initiates the commitment related to a norm to *Violated* that event e initiates also the commitment to perform the action described by the temporal proposition in the *active-sanction* attribute:

Axiom ActiveSanction
$HoldsAt(Norm(id, d, c, content, condition, e_{start}, a\text{-}sanction, p\text{-}sanction), t) \wedge$
$Initiates(e, Comm(Violated, d, c, content, condition, Norm, id), t) \wedge Happens(e, t) \rightarrow$
$Happens(AttCreateComm(d, c, a\text{-}sanction, TP(PTrue(), 1, 1, Exist), Sanction, id), t)$

Where the label *"Sanction"* as *source* attribute of the commitment means that the commitment has been generated as sanction of the norm with identifier *id* and the violation of this type of commitment will initiates neither another commitment nor a new power.

3.11.1 Norms expressed in terms of roles

Norms with specific agent as debtor and creditor have to holds in the initial state of the system, if fact for this work we will not introduce actions for creating norms at run-time. Given that in the design phase of a system it is impossible to know the name of the agents that will actually interact with the system, it is crucial to express norms in terms of the roles played by the agent in the interaction system. This can be obtained by means of suitable axioms (like the ones that we introduced to express power in terms of roles in Section 3.10.1) that initiate a certain norm when an agent enter a given role.

Given that the agents involved in a norm are the *debtor* and the *creditor* it may be necessary to define two axioms for the conditional initiation of a norm when those agents start to play a certain role. For example the norm that obliges the agent that play the role of *Boss* of an organization to pay x euro to the agents playing the role of *Employee* when the end of the month is reached can be expressed using the following axiom:

$E_1 =_{def} AssignRole(agent_x, agent1, Boss, Organization(1))$
$E_2 =_{def} AssignRole(agent_x, agent2, Employee, Organization(1))$
$TP1 =_{def} TP(Done(agent1, Pay(agent1, agent2, x)), t, t + 2, Exist)$
$TP2 =_{def} TP(PTrue(), 1, 1, Exist)$
$TP3 =_{def} TP(Done(agent1, Pay(agent1, agent2, x * 1.1)), t + 3, t + 5, Exist))$

$Happens(E_1, t_1) \wedge t_1 <= t \rightarrow$
$Initiates(E_2, Norm(agent1, agent2, TP1, TP2, Elapse(31), TP3,$
$\quad Power(InstAgent, ChangeReputation(agent1, -2))), t)$

where we assume that *ChangeReputation(agent,integer)* is an institutional action adding a given value to the reputation of an agent. The other axiom can

be obtained exchanging E_1 with E_2. It is also necessary to write two axioms to terminate the norm when an agent dismisses the role of *Boss* or of *Employee*.

4 Example

The Discrete Event Calculus Reasoner 1.0[3] as previously discussed is a tool that can be used to perform deduction, abduction and model finding starting from a domain description. Using such a tool and its deduction capabilities we verified that given a specification of an interaction system (consisting of the Discrete Event Calculus axioms, the OCeAN axioms introduced so far, and domain specific axioms), an initial situation, and a narrative of known event occurrences (that is a set of axioms that specify what happen in the system and when) there is only one model that satisfies them and represents how the system state will evolve.

The Discrete Event Calculus Reasoner 1.0 relies on various satisfiability (SAT) solver transforming formulas of first-order logic into formulas of the propositional calculus and by restricting the problem to a finite universe. Entailment in the propositional calculus is decidable but it is *NP-complete*. In practice (even if sometimes it is necessary to write more specific axioms) the specification of our examples (with over 10,000 variables) can be solved quite efficiently. In particular the axioms of our model can be quite forwardly transformed in a specification processable by the tool and introducing some adaptations due to certain tool limitations.

In this section we will present two examples. The first one shows a communicative interaction between two agents using the ACL defined in Section 3.7 and the temporal proposition and commitment axioms. The second example partially shows the dynamic evolution of a system designed for the execution of electronic auctions[4].

4.1 Example 1, usage of ACL

Here we describe an example of interaction where a *Seller* agent promises to deliver a product (a CD) to a *Buyer* agent if the *Buyer* will accept the *Seller*s request to commit itself to pay a certain amount of money (1 euro) for the product. Different possible evolution of the state of the interaction are possible on the basis of what the *Buyer* agent answers to the *Seller* request, and the various commitments created by the communicative acts will become fulfilled or violated on the basis of the actions performed by the *Seller* and *Buyer* agent.

For example given the following definitions,

[3] http://decreasoner.sourceforge.net/

[4] Files containing system specification for the Discrete Event Calculus Reasoner 1.0 that uses the axioms presented in this work and different simulations output obtained running those systems with different history of events are available at http://www.people.lu.unisi.ch/fornaran/code/Examples.html. Those simulations have been executed using Gygwin running on Windows XP on a desktop CORE2 DUO 1.8 GHz RAM 2 Gbyte

$TP1 =_{def} TP(Done(Seller, Deliver(Seller, Buyer, CD)), 4, 5, Exist)$
$TP2 =_{def} TP(Done(Buyer, Pay(Buyer, Seller, 1)), 6, 7, Exist))$
$TP3 =_{def} TP(Done(Buyer, ExchMsg(Agree, Buyer, Seller, TP2, TP1)), 2, 3, Exist)$

given the description of a system available in the file `ACL.e` and given the following history of events:

```
Happens(AttCreateTP(TP1),0).
Happens(AttCreateTP(TP2),0).
Happens(AttCreateTP(TP3),0).
Happens(ExchMsg(Promise,Seller,Buyer,TP1,TP3),1).
Happens(ExchMsg1(Request,Seller,Buyer,TP2,TP1,3),1).
Happens(ExchMsg(Agree,Buyer,Seller,TP2,TP1),2).
Happens(Deliver(Seller,Buyer,CD),4).
Happens(Pay(Buyer,Seller,1),6).
```

The output produced, that is, a simulation of the evolution of the state of the system is as follows. Fluents that become true are indicated with a plus sign ("+") and fluents that become false are indicated with a minus sign ("−"):

```
0
Happens(AttCreateTP(TP3), 0).
Happens(AttCreateTP(TP1), 0).
Happens(AttCreateTP(TP2), 0).
Happens(Elapse(0), 0).
1
+ETP(TP3, Undef).
+ETP(TP1, Undef).
+ETP(TP2, Undef).
+TP3().
+TP1().
+TP2().
Happens(ExchMsg(Promise, Seller, Buyer, TP1, TP3), 1).
Happens(AttCreateComm(Seller, Buyer, TP1, TP3), 1).
Happens(ExchMsg1(Request, Seller, Buyer, TP2, TP1, 3), 1).
Happens(AttCreatePrecomm(Buyer, Seller, TP2, TP1, 3), 1).
Happens(Elapse(1), 1).
```

Axioms `TP1` and `ETP0` creates the temporal propositions and the evaluated temporal proposition with *value Undef*. Axiom `A1` generate the event *Elapse(t)* for every value of *t*. The event of performing a promise communicative act generates, thanks to `Axiom Promise`, the event for creating a new commitment, whereas the request communicative act generates, thanks to `Axiom Request` a new precommitment that could be in the following accepted or refused by the receiver agent.

```
2
+Comm(Cond, Seller, Buyer, TP1, TP3).
```

```
+Precomm(Active, Buyer, Seller, TP2, TP1, 3).
Happens(ExchMsg(Agree, Buyer, Seller, TP2, TP1), 2).
Happens(AttAccceptPrecomm(Buyer, Seller, TP2, TP1), 2).
Happens(AttCreateComm(Buyer, Seller, TP2, TP1), 2).
Happens(Elapse(2), 2).
```

Axioms C01 and P1 initiates the commitment and precommitment fluents. The event of accepting the request, thanks to Axiom Agree, generates the event of accepting the precommitment generated by the previous request, the acceptance of a precommitment has also effect to attempt to create a new commitment (thanks to axiom P2).

```
3
-Precomm(Active, Buyer, Seller, TP2, TP1, 3).
+Comm(Cond, Buyer, Seller, TP2, TP1).
+Done(Buyer, ExchMsg(Agree, Buyer, Seller, TP2, TP1)).
-ETP(TP3, Undef).
+ETP(TP3, True).
-Comm(Cond, Seller, Buyer, TP1, TP3).
+Comm(Pending, Seller, Buyer, TP1, TP3).
Happens(Elapse(3), 3).
```

The *AttAccceptPrecomm()* terminates the precommitment (axiom P3) and the *AttCreateComm()* creates a *Cond* commitment (axiom C01). The performance of the agree communicative act initiates the *Done()* fluent (axiom Dn), transforms the *Undef ETP* having as *prop* the performance of the agree act into a *True ETP* (axioms ETPE5, ETP1), and transforms the *Cond* commitment between the *Seller* and the *Buyer* in a *Pending* one (axioms C1, C5).

```
4
Happens(Deliver(Seller, Buyer, CD), 4).
Happens(Elapse(4), 4).
5
+Done(Seller,Deliver(Seller,Buyer,CD)).
-ETP(TP1, Undef).
+ETP(TP1, True).
-Comm(Cond, Buyer, Seller, TP2, TP1).
+Comm(Pending, Buyer, Seller, TP2, TP1).
-Comm(Pending, Seller, Buyer, TP1, TP3).
+Comm(Fulfilled, Seller, Buyer, TP1, TP3).
Happens(Elapse(5), 5).
```

The delivery of the CD initiates the *Done()* fluent (axiom Dn), transforms the *Undef ETP* having as *prop* the deliver of the product into a *True ETP* (axioms ETPE5, ETP1), transforms the *Cond* commitment between the *Buyer* and the *Seller* in a *Pending* one (axioms C1, C5), and transforms the *Pending* commitment between the *Seller* and the *Buyer* into a *Fulfilled* commitment (axioms C3, C5).

```
6
Happens(Pay(Buyer, Seller, 1), 6).
Happens(Elapse(6), 6).
7
+Done(Buyer,Pay(Buyer,Seller,1)).
-ETP(TP2, Undef).
+ETP(TP2, True).
-Comm(Pending, Buyer, Seller, TP2, TP1).
+Comm(Fulfilled, Buyer, Seller, TP2, TP1).
Happens(Elapse(7), 7).
```

The payment for the CD initiates the *Done()* fluent (axiom `Dn`), transforms the *Undef ETP* having as *prop* such a payment into a *True ETP* (axioms `ETPE5`, `ETP1`), and transforms the *Pending* commitment between the *Buyer* and the *Seller* into a *Fulfilled* commitment (axioms `C3`, `C5`).

4.2 Example 2, a system for e-auctions

In this example we partially describe and test a system for the execution of electronic auctions, and simulate the evolution of its state for one possible event narrative. To exhaustively test the correctness of a given specification we would have to simulate the evolution of a system for every possible narrative and verify its correctness. As discussed in Section 3.1 the specification of this system consists of: (i) the set of fluents, events/actions and axioms that compose the *OCeAN* meta-model and presented in Section 3; (ii) the fluents, events/actions and axioms necessary for the specification of the institution of auctions (we introduce the fluent for representing auctions and the axioms `Auc1`, `Auc2`, `AucPower1`, and `AucNorm1` to create powers and norms); (iii) the fluents, events/actions and axioms specific to a certain type of auction like for example the English Auction; (iv) and finally the fluents, events/actions and axioms specific to a concrete system, like the fluent used to represent a product and the actions of paying or delivering it.

The *ientity* for representing auctions has been introduced in Section 3.9 as *Auction(id,astate,*t_{init}*)*. The effect of the *OpenAuction(id)* institutional action is defined introducing the following axioms:

Axiom Auc1
$HoldsAt(Auction(aid, Reg, t_{init}), t) \rightarrow Initiates(OpenAuction(id), Auction(aid, Open, t_{init}), t)$

Axiom Auc2
$HoldsAt(Auction(aid, Reg, t_{init}), t) \rightarrow Terminates(OpenAuction(id), Auction(aid, Reg, t_{init}), t)$

The initial state of the system is given by the following fluents that hold at time 0. The first represents the auction with *id* 01 and t_{init} 4, the other two represent the fact that the auction entity defines the roles of *Auctioneer* and *Participant*, and the last one represents the context for performing the specified *OpenAuction()* institutional action:

$Auction(01, Reg, 4)$
$RoleOf(Auctioneer, Auction(01, Reg, 4))$
$RoleOf(Participant, Auction(01, Reg, 4))$
$Context(OpenAuction(01, Reg, 4))$

The following axiom initiates the power, for the agent playing the role of *Auctioneer*, to declare open the auction for which it is playing that role, if at least two agents are already playing the role of *Participant*:

Axiom AucPower1
$\exists\, t_{init}\ Happens(AssignRole(agent1, agent3, Participant, Auction(aid, Reg, t_{init})), t_1) \wedge t_1 \leq t \wedge$
$Happens(AssignRole(agent1, agent4, Participant, Auction(aid, Reg, t_{init})), t_2) \wedge t_2 \leq t \wedge$
$Happens(AssignRole(agent1, agent2, Auctioneer, Auction(aid, Reg, t_{init})), t) \rightarrow$
$Happens(Empower(agent1, agent2, OpenAuction(aid)), t)$

Given the following definitions of temporal propositions:
$TP1 =_{def} TP(Done(OpenAuction(01)), t_{init}, t_{init} + 1, Exist)$
$TP2 =_{def} TP(PTrue(), 1, 1, Exist))$
$TP3 =_{def} TP(Done(Pay(Bob, AuctionHouse, 1)), t_{init} + 2, t_{init} + 3, Exist)$

the following axioms initiates the norm that creates the obligation for the agent playing the role of *Auctioneer* to declare open the auction when t_{init}-1 is elapsed:

Axiom AucNorm1
$Initiates(AssignRole(agent, agent1, Auctioneer, Auction(aid, Reg, t_{init})),$
$Norm(id, agent1, AuctionHouse, TP1, TP2, Elapse(t_{init}\text{-}1), TP3,$
$\quad Power(AuctionHouse, DecTrust(agent1)), t)$

The description of the system is available in the file `Auction.e`. Given the following history of events:

```
Happens(AttCreateTP(TP1), 0).
Happens(ExchMsgD(Declare,IntSystem,Bob,
        AssignRole(IntSystem,Bob,Auctioneer,Auction(01,Reg,4))),1).
Happens(ExchMsgD(Declare,IntSystem,Carl,
        AssignRole(IntSystem,Carl,Participant,Auction(01,Reg,4))),1).
Happens(ExchMsgD(Declare,IntSystem,Luke,
        AssignRole(IntSystem,Luke,Participant,Auction(01,Reg,4))),1).
Happens(ExchMsgD(Declare,Bob,Carl,OpenAuction(01)),4).
```

the output produced, that is, a simulation of the evolution of the state of the system is as follows. Fluents that become true are indicated with a plus sign ("+") and fluents that become false are indicated with a minus sign ("−"):

```
0
Happens(Elapse(0), 0).
Happens(AttCreateTP(TP1), 0).
```

```
1
Happens(Elapse(1), 1).
Happens(ExchMsgD(Declare, IntSystem, Bob,
          AssignRole(IntSystem, Bob, Auctioneer, Auction(01, Reg, 4))), 1).
Happens(ExchMsgD(Declare, IntSystem, Carl,
          AssignRole(IntSystem, Carl, Participant, Auction(01, Reg, 4))), 1).
Happens(ExchMsgD(Declare, IntSystem, Luke,
          AssignRole(IntSystem, Luke, Participant, Auction(01, Reg, 4))), 1).
Happens(AssignRole(IntSystem, Bob, Auctioneer, Auction(01, Reg, 4)), 1).
Happens(AssignRole(IntSystem, Carl, Participant, Auction(01, Reg, 4)), 1).
Happens(AssignRole(IntSystem, Luke, Participant, Auction(01, Reg, 4)), 1).
Happens(Empower(IntSystem, Bob, OpenAuction(01)), 1).
+ETP(TP1, Undef). +ETP(TP3, Undef).
+TP1(). +TP3).
```

Thanks to `Axiom Decl` the declared institutional actions happen (given that their contexts and the right powers hold). Moreover thanks to `Axiom AucPower1` a certain *Empower()* institutional action happens. Axioms `TP1` and `ETP0` initiates the temporal propositions and the evaluated temporal propositions.

```
2
Happens(Elapse(2), 2).
+HasRole(Bob, Auctioneer, Auction(01, Reg, 4)).
+HasRole(Carl, Participant, Auction(01, Reg, 4)).
+HasRole(Luke, Participant, Auction(01, Reg, 4)).
+Power(Bob, OpenAuction(01)).
+Norm(N1, Bob, AuctionHouse, TP1, TP2, Elapse(3), TP3,
                          Power(AuctionHouse, DecTrust(Bob))).
```

Thanks to `Axiom Role1` the effects of the *AssignRole()* action takes place and the agents start to hold the role of *Auctioneer* or of *Participant*. Thanks to `Axiom Powerl` the power for agent *Bob* to open the auction where it play the role of *Auctioneer* starts to hold. Thanks to `Axiom AucNorml` the norm that obliges the agent playing the role of *Auctioneer* for a certain auction to declare open such an auction is created.

```
3
Happens(Elapse(3), 3).
Happens(AttCreateComm(Bob, IntSystem, TP1, TP2, SNorm, N1), 3).
```

Thanks to `Axiom ActivateNorm` when the e_{start} event of a norm happens the norm becomes active and the *AttCreateComm()* action to create the related commitment happens.

```
4
Happens(Elapse(4), 4).
```

```
Happens(ExchMsgD(Declare, Bob, Carl, OpenAuction(01)), 4).
Happens(OpenAuction(01), 4).
+Comm(Pending, Bob, AuctionHouse, TP1, TP2, SNorm, N1).
```

Given that the right context and the power hold thanks to `Axiom Decl` the declared *OpenAuction()* action happens and thanks to axiom `C03` a *Pending* commitment is initiated.

```
5
-Auction(01, Reg, 4).
+Auction(01, Open, 4).
-ETP(TP1, Undef).
+ETP(TP1, True).
-Comm(Pending, Bob, AuctionHouse, TP1, TP2, SNorm, N1).
+Comm(Fulfilled, Bob, AuctionHouse, TP1, TP2, SNorm, N1).
```

Thanks to axioms `Auc1` and `Auc2` an *Open* auction is initiated and the obligation to open the auction between time 4 and time 5 generated by norm *N1*, which is represented by means of the commitment, is *Fulfilled* (axiom `ETPE1` and `C3`).

5 Conclusion and Future Research Directions

Using the Event Calculus, we have presented a formal specification of the *OCeAN* meta-model for open artificial institutions. With respect to other formalisms with well-understood formal semantics, the main advantage of the Event Calculus is that it allows for the execution of the formal specification of a system to test its correctness, while retaining the expressiveness of first order logic.

Our formal definition of *OCeAN* allows one to define and test the specification of an open interaction system, conceived as a set of artificial institutions. Given that all components of *OCeAN* are explicitly defined in logic, in principle it is possible to design agents that can interact with different OCeAN-based interaction systems without being reprogrammed. Other possible applications of our meta-model are the specification and monitoring of contracts by means of commitments, and the flexible specification of protocols using norms and the library of communicative acts.

Indeed, *OCeAN* is still incomplete under a number of respects. For example, it will be important to understand how a new institution may be defined using, or inheriting concepts from, other institutions. It is also important to understand whether and how the meta-model has to be enriched to be used for different types of applications, like for example for *Virtual Enterprises* [6], for the flexible specification of system for knowledge sharing [11], or for collaborative environments. Moreover, it will be necessary to analyse and formally specify different types of institutional powers: for example, it may be useful to distinguish between the power to perform an action as many time as needed from the

power to perform it only once, and to analyse the type of power required by passive sanctions.

In particular we plan to continue our research with a focus on mixing human and artificial agents in hybrid multiagent systems within a given organizational structure, as an environment for complex collective activities. The practical design and implementation of such systems involves the study of two interconnected problems. The first is to develop a framework that allows one to design software agents able to interact with different open systems (for example different running auctions) without being reprogrammed; this requires that software agents are able to access a formal description of the interaction system, including the context of the interaction and the norms regulating the system, and to reason at runtime on how to reach their goals. The second problem is how to use and extend artificial institutions to represent the human organizational structures within which interactions take place, assuming that the software agents participating in the interaction have sufficient reasoning capabilities.

References

[1] M. Alberti, M. Gavanelli, E. Lamma, F. Chesani, P. Mello, and P. Torroni. A logic based approach to interaction design in open multi-agent systems. In *WETICE '04*, pages 387–392, Washington, DC, USA, 2004. IEEE Computer Society.

[2] J. L. Arcos, M. Esteva, P. Noriega, J. A. Rodríguez-Aguilar, and C. Sierra. Engineering open evironments with electronic institutions. *Engineering applications of artificial intelligence*, 18(2):191 – 204, 2005.

[3] A. Artikis, M. Sergot, and J. Pitt. Animated Specifications of Computational Societies. In C. Castelfranchi and W. L. Johnson, editor, *AAMAS 2002*, pages 535–542. ACM Press, 2002.

[4] K. Bach and R. M. Harnish. *Linguistic Communication and Speech Acts*. MIT Press, Cambridge, MA, 1979.

[5] Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen, editors. *Normative Multi-agent Systems, 18-23 March 2007*, volume 07122 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[6] H. L. Cardoso and E. C. Oliveira. Virtual enterprise normative framework within electronic institutions. In *ESAW 2004*, pages 14–32, 2004.

[7] H. Lopes Cardoso and E. Oliveira. Institutional reality and norms: Specifying and monitoring agent organizations. *International Journal of Cooperative Information Systems*, 16(1):67–95, 2007.

[8] O. Cliffe, M. De Vos, and J. A. Padget. Answer set programming for representing and reasoning about virtual institututions. In K. Inoue, K. Satoh, and F. Toni, editors, *CLIMA VII, Hakodate, Japan, May 2006*, volume 4371/2007 of *LNCS*, pages 60–79. Springer Berlin, 2006.

[9] O. Cliffe, M. De Vos, and J. Padget1. Specifying and reasoning about multiple institutions. In P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, and E. Matson, editors, *COIN II*, volume 4386/2007 of *LNCS*, pages 67–85. Springer Berlin, 2007.

[10] M. Colombetti and M. Verdicchio. An analysis of agent speech acts as institutional actions. In C. Castelfranchi and W. L. Johnson, editors, *AAMAS 2002*, pages 1157–1166, 2002.

[11] Virginia Dignum, Frank Dignum, and John-Jules Meyer. An agent-mediated approach to the support of knowledge sharing in organizations. *Knowl. Eng. Rev.*, 19(2):147–174, 2004.

[12] 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.

[13] A. D. H. Farrell, M. J. Sergot, M. Sallé, and C. Bartolini. Using the event calculus for tracking the normative state of contracts. *International Journal of Cooperative Information Systems (IJCIS)*, 14(2-3):99–129, 2005.

[14] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. M. Bradshaw, editor, *Software Agents*, chapter 14, pages 291–316. AAAI Press / The MIT Press, 1997.

[15] FIPA. Foundation for Intelligent Physical Agents (FIPA) Communicative Act Library Specification. http://www.fipa.org, 2002.

[16] N. Fornara and M. Colombetti. A commitment-based approach to agent communication. *Applied Artificial Intelligence an International Journal*, 18(9-10):853–866, 2004.

[17] N. Fornara and M. Colombetti. Specifying and enforcing norms in artificial institutions. In Guido Boella, Leon van der Torre, and Harko Verhagen, editors, *Normative Multi-agent Systems*, number 07122 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[18] N. Fornara and M. Colombetti. Formal specification of artificial institutions using the event calculus. Technical Report 5, Università della Svizzera italiana, April 2008.

[19] N. Fornara, F. Viganò, and M. Colombetti. Agent communication and artificial institutions. *Autonomous Agents and Multi-Agent Systems*, 14(2):121–142, April 2007.

[20] N. Fornara, F. Viganò, M. Verdicchio, and M. Colombetti. Artificial institutions: A model of institutional reality for open multiagent systems. *Artificial Intelligence and Law*, 16(1):89–105, March 2008.

[21] A. García-Camino, P. Noriega, and J. A. Rodríguez-Aguilar. Implementing norms in electronic institutions. In *AAMAS'05*, pages 667–673, New York, NY, USA, 2005. ACM Press.

[22] D. Grossi, H. Aldewereld, and F. Dignum. Ubi lex, ibi poena: Designing norm enforcement in e-institutions. In P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, and E. Matson, editors, *COIN II*, volume 4386 of *LNCS*, pages 101–114. Springer Berlin, 2007.

[23] H. L. A. Hart. *The Concept of Law.* Clarendon Press, Oxford, 1961.

[24] Carl Hewitt. Offices are open systems. *ACM Trans. Inf. Syst.*, 4(3):271–287, 1986.

[25] L. Kagal and T. Finin. Modeling Conversation Policies using Permissions and Obligations. In R. van Eijk, M. Huget, and F. Dignum, editors, *Developments in Agent Communication*, volume 3396 of *LNCS*, pages 123–133. Springer, 2005.

[26] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[27] F. López y López, M. Luck, and M. d'Inverno. A Normative Framework for Agent-Based Systems. In *Proceedings of the First International Symposium on Normative Multi-Agent Systems, Hatfield*, 2005.

[28] 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.A. Williams, editors, *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004), Whistler, Canada, June 2-5 2004*, pages 267–277, 2004.

[29] R. Miller and M. Shanahan. Some alternative formulations of the event calculus. In A. C. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond: Essay in Honour of Robert A. Kowalski, Part II*, volume LNCS 2408, pages 452–490. Springer, Berlin, 2002.

[30] E. T. Mueller. *Commonsense Reasoning.* Morgan Kaufmann, San Francisco, 2006.

[31] E. T. Mueller. Event calculus. In F. van Hermelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation.* Elsevier, Amsterdam, 2007.

[32] P. Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autnoma de Barcelona, 1997.

[33] J. R. Searle. *The construction of social reality*. Free Press, New York, 1995.

[34] M. Shanahan. *Solving the Frame Problem*. MIT Press, Cambridge, MA, 1997.

[35] M. Shanahan. The Event Calculus Explained. In M. J. Wooldridge and M. M. Veloso, editors, *Artificial Intelligence Today: Recent Trends and Developments*, volume LNCS 1600, pages 409–430. Springer, Berlin, 1999.

[36] M. Shanahan. An abductive event calculus planner. *Journal of Logic Programming*, 44(1-3):207–240, 2000.

[37] M. P. Singh. A social semantics for agent communication languages. In *Proceedings of IJCAI-99 Workshop on Agent Communication Languages*, pages 75–88, 1999.

[38] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.

[39] J. Vázquez-Salceda, V. Dignum, and F. Dignum. Organizing multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 11(3):307–360, Nov 2005.

[40] F. Viganò, N. Fornara, and M. Colombetti. An Event Driven Approach to Norms in Artificial Institutions. In O. Boissier, J. Padget, V. Dignum, G. Lindemann, E. Matson, S. Ossowski, J. Simao Sichman, and J. Vázquez-Salceda, editors, *COIN I*, volume LNAI 3913, pages 142–154. Springer Berlin, 2006.

[41] P. Yolum and M.P. Singh. Reasoning about commitment in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence*, 42:227–253, 2004.

[42] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.