Normative KGP Agents: A Preliminary Report

Fariba Sadri and Francesca Toni*
*Department of Computing,
Imperial College London, UK.
{fs,ft}@doc.ic.ac.uk

Kostas Stathis†
†Department of Computing,
City University London, UK.
kostas@soi.city.ac.uk

Abstract

We extend the logical model of agency known as the KGP model, to support agents with normative concepts, using obligations and prohibitions as examples. The proposed framework illustrates how to integrate normative concepts and roles within the KGP model in such a way that these concepts can evolve dynamically. Furthermore, we illustrate how these concepts can be combined with the existing capabilities of KGP agents in order to plan and react to changes in the environment. Our approach gives an executable specification of normative concepts that can be used directly for prototyping applications.

1 Introduction

Programmers that develop complex distributed systems based on autonomous agents often need to find ways of decentralising the functionality of the whole system by distributing responsibility to the parts, while still ensuring that interactions of the whole are coherent and coordinated. Whether we deal with autonomous robots that plan, a traffic or a file-sharing system, or any other similar application, it is becoming increasingly recognised that the resolution of the underlying problems lies with developing frameworks that are based on the notion of social agency (Huhns and Singh, 1998).

The basic idea behind the notion of social agency is to use abstractions from such diverse fields as sociology, computing science, organisational theory, and law in order to specify (Jones and Sergot, 1993) and then implement complex organisations of agents referred to as *artificial societies* (Padget, 2001). Such an implementation seeks to apply social agents in practical applications where the formation of open societies (Artikis and Pitt, 2001) are envisaged to be regulated by *norms*. The notion of a norm is important here in that member agents of a society (Toni and Stathis, 2002) must have the capability to reason with norms and they must be capable of communicating norms to other member agents. The problem then reduces to resolving the issue of how to develop normative autonomous agents (Dignum, 1999) for supporting practical applications based on artificial societies.

To develop normative agents programmers often rush into implementing the rules governing an artificial society without properly understanding the subtleties that underlie their specification. Motivated by this observation a number of researchers, e.g. see (Castelfranchi et al., 1999; Jones and Sergot, 1996; Carabelea et al., 2004), are seeking to understand the modalities required to specify the norms of an artificial society as a separate issue from their implementation. Deontic concepts such as obligation, prohibition, permission, rights, power, and entitlement, are currently being scrutinised and analysed in detailed formal frameworks that have been produced as a result. Much of the work in this effort, however, can only be used as a guide to implementations, as it abstracts away from the computational characterisation of the resulting specifications and, more importantly, from the way these normative concepts are to be used during the operation of the agent in an artificial social environment.

Motivated by this last observation, this work seeks to complement the specification effort previously described by presenting a framework that illustrates how normative concepts, such as obligations and prohibitions, can be used by an agent while it reasons, reacts, plans, and communicates in the context of an artificial society. We develop this framework by building upon existing work with the KGP (Knowledge, Goals, Plans) model of agency (Kakas et al., 2004b) which we have successfully implemented in the prototype agent platform PROSOCS (Stathis et al., 2004). The contribution of the work is to show how to extend the KGP model with normative concepts, thus providing a framework where we can develop agents who reason about norms that govern, not only their own behaviour, but also the behaviour of other agents. One major advantage of

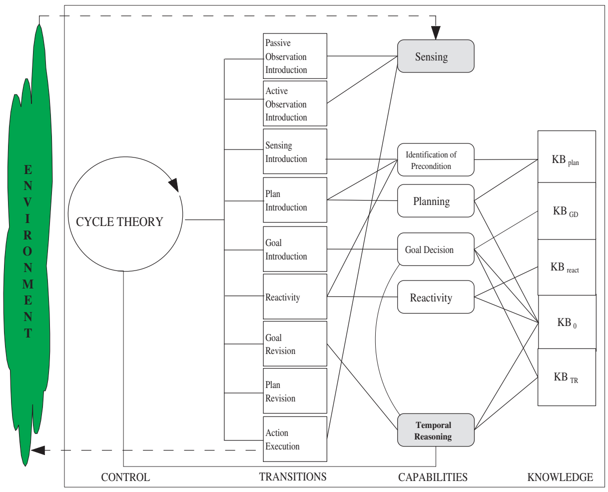

Figure 1: The KGP Model of Agency.

our proposed approach is that the extension to the KGP model is smooth, without requiring new capabilities, but simply extending the existing knowledge bases with which the agents reason when they operate. Another distinguishing feature of our approach that we consider as another major advantage is that the normative rules that we propose can be looked at in many different ways: they are executable specifications, they are directly implementable and, within the declarative and operational model of the KGP agents, they force agents to exhibit the expected behaviour, conformant with the specification.

We structure the paper as follows. In the next section we summarise the main features of the KGP model. Then, in the following section we outline how to extend the current model so that we can accomodate normative concepts such as obligations. We then exemplify the extended model with a scenario, showing how an agent can use the specifications to produce norm-governed baheaviour. Related work is discussed in the penultimate section, while in the final section we summarise the main features of the work and outline our plans for future work.

2 The KGP Model

Here we briefly summarise the KGP model for agents, which is depicted in Figure 1. We focus on the components relevant to this paper, see (Kakas et al., 2004b; Bracciali et al., 2004) for any additional details.

The model relies upon:

- an internal (or mental) state,
- a set of reasoning capabilities, supporting planning, temporal reasoning, identification of preconditions of actions, reactivity and goal decision,
- a sensing capability,
- a set of transition rules, defining how the state of the agent changes, and defined in terms of the above capabilities,
- a set of selection functions, to provide appropriate inputs to the transitions,
- a cycle theory, for deciding which transitions should be applied when, and defined using the selection functions.

Internal state. This is a tuple

$$\langle KB, Goals, Plan, TCs \rangle,$$

where:

- **KB** describes what the agent knows of itself and the environment and consists of separate modules supporting the different reasoning capabilities, including
  - KB_{plan}, for Planning,
  - KB_{pre}, for the Identification of Preconditions of actions,
  - KB_{react}, for Reactivity, and
  - KB_0, for holding the (dynamic) knowledge of the agent about the external world in which it is situated (including past communications), and perceived through its sensing capability.

  Syntactically, KB_{plan} and KB_{react} are abductive logic programs with constraint predicates (see (Bracciali et al., 2004)), and KB_{pre} is a logic program.

- **Goals** is a set of properties that the agent wants to achieve, each one explicitly time-stamped by a time variable. Goals may also be equipped with a temporal constraint (given in TCs) binding the time variable and constraining when the goals are expected to hold. Goals may be *mental* or *sensing*. Both can be observed to hold (or not to hold) via the Sensing capability. In addition, mental goals can be brought about actively by the agent by its Planning capability and its actions.

- **P**lan is a set of actions scheduled in order to satisfy goals. Each is explicitly time-stamped by a time variable and possibly equipped with a temporal constraint (given in TCs), similarly to $Goals$, but constraining when the action should be executed. Actions are partially ordered, via their temporal constraints. Each action is also equipped with the preconditions for its successful execution, determined by the Identification of Preconditions capability. Actions may be *physical*, *communicative*, or *sensing*.

- TCs is a set of constraint atoms (referred to as *temporal constraints*) in some given underlying constraint language. We assume that the constraint predicates include $<, \leq, >, \leq, =, \neq$. These constraints specify when goals are to hold and when actions are to be executed, and they are extended and instantiated as the agent operates.

Reasoning capabilities. These include:

- Planning, which generates partial plans for sets of goals. It provides (temporally constrained) sub-goals and actions designed for achieving the input goals.

- Reactivity, which reacts to perceived changes in the environment, by replacing (some) goals in $Goals$ and actions in $Plan$ with (possibly temporally constrained) goals and actions.

- Identification of Preconditions for action execution.

Sensing capability. This links the agent to its environment, by allowing to observe that properties hold or do not hold, and that other agents have executed actions in the past. It also allows agents to receive communication from other agents.

Transitions. The state of an agent evolves by applying transition rules, which employ capabilities. The transitions include:

- *Passive Observation Introduction* (POI) changes KB_0 by introducing unsolicited information coming from the environment or communications received from other agents. It calls the Sensing capability.

- *Plan Introduction* (PI) changes part of the $Goals$ and $Plan$ and TCs of a state, according to the output of the Planning capability. This transition uses also the Identification of Preconditions capability, in order to equip each planned action A with the set of preconditions for the successful execution of A.

- *Reactivity* (RE) is responsible for updating the current state of the agent by adding the goals and actions, together with any additional temporal constraints, returned by the Reactivity capability. As with PI, this transition too uses the Identification of Preconditions capability, in order to equip each new action A with the set of preconditions for the successful execution of A.

- *Action Execution* (AE) is responsible for executing all types of actions, thus changing the KB_0 part of KB by adding evidence that actions have been executed. It calls the Sensing capability for the execution of sensing actions.

Cycle. The behaviour of an agent is given by the application of transitions in sequences, repeatedly changing the state of the agent. These sequences are not determined by fixed cycles of behaviour, as in conventional agent architectures, but rather by reasoning with cycle theories. These are logic programs with priorities (see (Kakas et al., 2004a)), defining preference policies over the order of application of transitions, which may depend on the environment and the internal state of a agent.

In the remainder of this section, we give some details on the state of agents in the original KGP model, to provide the necessary background to incorporate normative concepts.

Goals. A goal is a *timed fluent literal* $l[t]$, where l refers to a (positive or negative) property that the agent wants to hold and t is the *time* of the goal, namely a variable, implicitly existentially quantified within the overall state of the agent. An example is $has\_driving\_licence(ag_1, t_1)$, indicating that agent ag_1 wants to have a driving licence at time t_1. This time may be constrained by TCs in the state, e.g. TCs may contain $10 < t_1 < 20$.

Actions. An action is a *timed operator* $a[t]$, where a refers to the operator of the action, and t is the *execution time* of the action, namely a variable, implicitly existentially quantified within the overall state of the agent. An example is $pay\_fine(ag_1, t_2)$, indicating that agent ag_1 wants to perform an act of paying a fine at time t_2. Again, this time may be constrained by TCs in the state, e.g. TCs may contain $t_2 < t_1$.

KB_0. This records the actions which have been executed (by the agent or by others) and their time of execution as well as the properties (i.e. fluents and their negation) which have been observed, possibly concerning other agents, and the time of the observation. Formally, KB_0 contains assertions of the form:

- $executed(a[t], \tau)$ where $a[t]$ is a timed operator and τ is a time constant, meaning that action a has been executed at time $t = \tau$, by the agent holding the KB_0.
- $observed(ag', a[ag, \tau], \tau')$ where τ is a time constant, meaning that action a has been executed at time τ by the agent ag, and this has been observed by the agent ag' (different from ag) holding the KB_0.
- $observed(l[t], \tau)$ where $l[t]$ is a timed fluent literal and τ is a time constant, meaning that the property l has been observed to hold at time $t = \tau$, by the agent holding the KB_0.

KB_{plan} and Planning capability. KB_{plan}, KB_{react}, and KB_{pre} are all specified within the framework of the event calculus (EC) for reasoning about actions, events and changes (Kowalski and Sergot, 1986). Below, we give the abductive logic program KB_{plan} and the logic program KB_{pre}.

In a nutshell, the EC allows to write meta-logic programs which "talk" about object-level concepts of *fluents*, *events* (that we interpret as *action operations*), and *time points*. The main meta-predicates of the formalism are: $holds\_at(F, T)$ (a fluent F holds at a time T), $clipped(T_1, F, T_2)$ (a fluent F is clipped - from holding to not holding - between times T_1 and T_2), $declipped(T_1, F, T_2)$ (a fluent F is declipped - from not holding to holding - between times T_1 and T_2), $initially(F)$ (a fluent F holds from the initial time, say time 0), $happens(O, T)$ (an operation O happens at a time T), $initiates(O, T, F)$ (a fluent F starts to hold after an operation O at time T) and $terminates(O, T, F)$ (a fluent F ceases to hold after an operation O at time T). Roughly speaking, in a planning setting the last two predicates represent the cause-effects links between operations and fluents in the modelled world. We will also use a meta-predicate $precondition(O, F)$ (the fluent F is one of the preconditions for the executability of the operation O).

KB_{plan} consists of domain-independent rules and domain-dependent rules. The basic domain-independent rules are

$$
\begin{aligned}
& holds\_at(F, T_2) \leftarrow \\
& \qquad happens(O, T_1) \wedge \\
& \qquad initiates(O, T_1, F) \wedge \\
& \qquad T_1 < T_2 \wedge \\
& \qquad not\ clipped(T_1, F, T_2) \\
& holds\_at(\neg F, T_2) \leftarrow \\
& \qquad happens(O, T_1) \wedge \\
& \qquad terminates(O, T_1, F) \wedge \\
& \qquad T_1 < T_2 \wedge \\
& \qquad not\ declipped(T_1, F, T_2) \\
& holds\_at(F, T) \leftarrow \\
& \qquad initially(F) \wedge \\
& \qquad 0 \leq T \wedge \\
& \qquad not\ clipped(0, F, T) \\
& holds\_at(\neg F, T) \leftarrow \\
& \qquad initially(\neg F) \wedge \\
& \qquad 0 \leq T \wedge \\
& \qquad not\ declipped(0, F, T) \\
& clipped(T_1, F, T_2) \leftarrow \\
& \qquad happens(O,\ T) \wedge \\
& \qquad terminates(O, T, F) \wedge \\
& \qquad T_1 \leq T < T_2 \\
& declipped(T_1, F, T_2) \leftarrow \\
& \qquad happens(O,\ T) \wedge \\
& \qquad initiates(O, T, F) \wedge \\
& \qquad T_1 \leq T < T_2
\end{aligned}
$$

The *domain-dependent rules* define $initiates$, $terminates$, and $initially$, e.g.

$$
\begin{aligned}
& initiates(go(X, L_1, L_2), T, at(X, L_2)) \leftarrow \\
& \qquad holds\_at(mobile(X), T) \\
& initiates(go(X, L_1, L_2), T, free(L_1)) \leftarrow \\
& \qquad holds\_at(mobile(X), T) \wedge \\
& \qquad L_1 \neq L_2 \\
& terminates(go(X, L_1, L_2), T, at(X, L_1) \leftarrow \\
& \qquad holds\_at(mobile(X), T) \wedge \\
& \qquad L_1 \neq L_2 \\
& terminates(go(X, L_1, L_2), T, free(L_2) \leftarrow \\
& \qquad holds\_at(mobile(X), T) \wedge \\
& \qquad L_1 \neq L_2 \\
& initially(at(ag_1, (1, 1)))
\end{aligned}
$$

Namely, the operation go from one location L_1 to some other location L_2 initiates the agent (robot) X being at location L_2 and location L_1 being free and terminates X being at location L_1 and location L_2 being free, provided that X is mobile. Moreover, some agent ag_1 is initially at location $(1,1)$. The conditions for the rules defining $initiates$ and $terminates$ can be seen as preconditions for the effects of the operator go to take place. Preconditions for the executability of operators are specified within KB_{pre}, which contains a set of rules defining the predicate $precondition$, e.g.

$$precondition(go(X, L_1, L_2), at(X, L_1))$$
$$precondition(go(X, L_1, L_2), free(L_2))$$

namely the preconditions of the operator $go(X, L_1, L_2)$ are that X is at the initial location L_1 and that location L_2, where X is moving to, is free.

In order to accommodate planning we will assume that the domain-independent part also contains the rules:

$$happens(O, T) \leftarrow assume\_happens(O, T)$$

i.e. actions (by the agent itself) can be made to happen simply by assuming them (by abduction). Note that the abduction of atoms $assume\_happens(a[ag], t)$ by agent ag amounts to planning to execute the corresponding action $a[ag, t]$ by agent ag, to achieve a goal initiated by that action. Similarly, atoms $holds\_at(l, t)$ in the event calculus language correspond to goals $l[t]$ in the state of agents.

The domain-independent part of KB_{plan} also contains the following domain-independent integrity constraints:

$$holds\_at(F, T) \wedge holds\_at(\neg F, T) \Rightarrow false$$

$$\begin{array}{l} assume\_happens(O, T) \wedge \\ precondition(O, P) \Rightarrow holds\_at(P, T) \end{array}$$

$$\begin{array}{l} assume\_happens(O, T) \wedge \\ not\ executed(O, T) \wedge \\ time\_now(T') \Rightarrow T > T' \end{array}$$

namely a fluent and its negation cannot hold at the same time, when assuming (planning) that some action will happen, we need to enforce that each of its preconditions hold and that this action will be executable in the future.

To allow agents to draw conclusions from the contents of KB_0, which represent the "narrative" part of the agent's knowledge, the following *bridge rules* are also amongst the domain independent rules of KB_{plan}:

$$\begin{array}{l}
clipped(T_1, F, T_2) \leftarrow \\
\qquad observed(\neg F[_], T) \wedge \\
\qquad T_1 \leq T < T_2 \\
declipped(T_1, F, T_2) \leftarrow \\
\qquad observed(F[_], T) \wedge \\
\qquad T_1 \leq T < T_2 \\
holds_at(F, T_2) \leftarrow \\
\qquad observed(F[_], T_1) \wedge \\
\qquad T_1 \leq T_2 \wedge \\
\qquad not\ clipped(T_1, F, T_2) \\
holds_at(\neg F, T_2) \leftarrow \\
\qquad observed(\neg F[_], T_1) \wedge \\
\qquad T_1 \leq T_2 \wedge \\
\qquad not\ declipped(T_1, F, T_2) \\
happens(O, T) \leftarrow executed(O, T) \\
happens(O, T) \leftarrow observed(_, O[T'], T)
\end{array}$$

Note that we assume that the value of a fluent literal is changed according to observations only from the moment the observations are made, and actions by other agents have effects only from the time observations are made that they have been executed, rather than by the execution time itself. These choices are dictated by the rationale that observations can only be considered and reasoned upon from the moment the planning agent makes them.

Taken KB_{plan} and a goal, the planning capability returns a plan, namely a set of actions and sub-goals, such that by assuming them the goal can be proven to hold in KB_{plan}. For a formal definition, see (Bracciali et al., 2004).

KB_{react} and Reactivity capability. The Reactivity capability supports the reasoning capability of reacting to stimuli from the external environment as well as to decisions taken while planning. The capability introduces goals and actions in order to react to some observation recorded in (the KB_0 part of) the given KB or to some goals in $Goals$ and actions in

$Plan$. The reactivity capability allows us to incorporate condition-action rule behaviour, plan repair, policy-based communication. Moreover, as we will see later, this capability allows us to endow agents with norm-conformant behaviour.

The knowledge base KB_{react} represents the knowledge required for reactivity and it is suitably adopted as an extension of the knowledge base KB_{plan}, by adding domain-dependent *reactive rules*, of the form

$$Body \Rightarrow Reaction \wedge TCs$$

where

- $Body$ is a non-empty conjunction of items of the form
  - $observed(l[\_], T)$, where l is a fluent literal,
  - $executed(a[T'], T'')$, where $a[T']$ is a timed action operator,
  - $observed(ag, a[T'], T'')$, where $a[T']$ is a timed action operator and ag is the agent holding KB_{react},
  - $holds\_at(l, T')$, where $l[T']$ is a timed fluent literal,
  - $happens(a, T')$, where $a[T']$ is a timed action operator, and
  - temporal constraints;
- $Reaction$ is either $holds\_at(p, T)$, p being a fluent literal, or $assume\_happens(a, T)$, a being an action, and
- TCs are temporal constraints.

All variables in $Body$ are implicitly universally quantified over the whole implication. All variables in $Reaction \wedge TCs$ not occurring in $Body$ are implicitly existentially quantified on the righthand side of the implication.

Intuitively, a reactive rule $Body \Rightarrow Reaction \wedge TCs$ is to be interpreted as follows: if (some instantiation of) all the observations in $Body$ hold in KB_0 and (some corresponding instantiation of) all the remaining conditions in $Body$ hold, then (the appropriate instantiation of) $Reaction$, with associated (the appropriate instantiation of) the temporal constraints TCs, should be added to $Goals$ or $Plan$ (depending on the nature of $Reaction$).

Taken KB_{react}, the reactive capability returns a set of actions and goals and temporal constraints, such that all the reactive rules in KB_{react} can be proven to hold in KB_{plan} extended with the state resulting from reactivity. The actions will then have to be part of any plan by the agent, and the goals will need to be planned for so that they hold.

3 Extending KGP with Normative Concepts

So far we have summarised the main features of the KGP model for agency. Here we will show how this model can incorporate normative reasoning, simply by adapting the event calculus used originally in the KGP model, for planning and reactivity. The type of scenario that motivates our work is that of an artificial society of agents relying upon an organisation and division of tasks. We will interpret responsibility for tasks of an agent in terms of the roles an agent has been assigned to play in a social environment. Roles are associated with obligations, permission and prohibition, used to define what is expected of the agent when playing a role. Rather than developing or deploying any fully-fledged normative theory, we concentrate on a simple theory (for "responsible" agents) and explore its interaction with the existing operations of KGP agents and the provable conformance of the agents to this theory.

Reactivity is used to make agents aware of their obligations and prohibitions, depending on their observations and the environment in which they are situated. It is also used to force the agents to conform to any relevant normative rules. Planning is used to plan activities that would allow agents to achieve their obligations while avoiding their prohibitions, if possible. As we will see, we define the normative concepts in such a way that agents may have information about other agents' obligations and prohibitions, and thus possibly exploit this information towards the achievement of their objectives.

3.1 Actions and Fluents

In general obligations and prohibitions will be on performing actions or bringing about fluents. Within obligations and prohibitions we will represent actions as terms of the form

$$act(Act, Actor, Parameters).$$

Act names the action, *Actor* is the agent to carry out the act, and *Parameters* is the set of attributes that further specify the action. For instance, the term:

$$act(pay\_fine, ag_1, ''W129FGC'', 60)$$

represents the action of agent ag_1 having to pay a fine of 60 pounds for a car with registration number W129FGC.

For representational convenience, we will often abuse terminology and write $act(Act, Actor)$ if we are only interested in the action and the agent of the action.

We shall represent fluents within obligations and prohibitions as terms of the form:

$$fluent(Fluent, Actor, Parameters).$$

Fluent is the name of the fluent, *Actor* is the agent to bring about *Fluent*, and, as before, *Parameters* represents the rest of the attributes describing the *Fluent*. For instance, the term:

$$fluent(has\_driving\_licence, ag_1,''ST340578KS'')$$

represents the fluent that agent ag_1 has a driving licence with number ST340578KS.

As before, we will often abuse terminology and write $fluent(Fluent, Actor)$ if we are only interested in the fluent and the agent.

3.2 Roles

In our model the agents' responsability for tasks and assignemnt of obligations and prohibitions is determined by the *roles* the agents play in their social environment. Therefore, we incorporate within the KGP model roles assigned to agents. We shall use the fluent:

$$role(Agent, Role)$$

to state when a specific *Role* is assigned to a specific *Agent*. $role/2$ fluents are initiated and terminated by means of event calculus $initiates/3$ and $terminates/3$ predicates. For instance, the addition of the following domain-specific rule for $initiates/3$:

$$\begin{array}{l} initiates(assign(Y, X, t\_w(W, F, T), \\ \qquad T', \\ \qquad role(X, t\_w(W, F, T))) \end{array}$$

allows us to conclude that after the occurrence of an action of the form

$$assign(ag_1, ag_2, t\_w(chelsea, 9, 17))$$

ag_2 plays the role of traffic warden (t_w) in $chelsea$ between times 9 and 17.

Depending on the ontology, we can qualify the assignment of roles, e.g. by means of a rule such as

$$\begin{array}{l} initiates(assign(Y, X, t\_w(W, F, T), \\ \qquad T', \\ \qquad role(X, t\_w(W, F, T)) \\ \qquad) \leftarrow \\ authorised(act(assign(Y, X, t\_w(W, F, T), Y), T') \end{array}$$

which relies upon a notion of "authorisation". We will not address this notion further in this paper.

Note that roles of agents might change as time progresses and will be initiated or terminated as a result of events happening in the social environment in which an agent is situated.

3.3 Obligations

We represent obligations as atoms of the form:

$$\begin{array}{l} obliged(act(Act, Actor), T, TCs) \\ obliged(fluent(Fluent, Actor), T, TCs) \end{array}$$

We read atoms of the first kind as follows: there is an obligation on the agent *Actor* to bring about the action specified by *Act* (with the appropriate *Parameters*) at a time *T* that satisfies the temporal constraints specified in *TCs*. We read atoms of the second kind as follows: there is an obligation on the agent *Actor* to bring about the fluent *Fluent* to hold at a time *T* that satisfies the temporal constraints specified in *TCs*.

Obligations held by individual agents are generated by means of reactive rules in KB_{react} in the knowledge base of the agents. These reactive rules take the general forms:

$$\begin{array}{l} holds\_at(role(Actor, Role), T_{now}) \wedge \\ Conditions(Actor, T_{now}) \Rightarrow \\ obliged(act(Act, Actor), T_{next}, TCs)) \end{array}$$

$$\begin{array}{l} holds\_at(role(Actor, Role), T_{now}) \wedge \\ Conditions(Actor, T_{now}) \Rightarrow \\ obliged(fluent(Fluent, Actor), T_{next}, TCs)) \end{array}$$

The rules above can be read as follows: if an *Actor* is playing a *Role* at time T_{now} and the *Conditions* hold in the knowledge base of the *Actor* at T_{now}, the *Actor* must fulfill the obligation of making the *Act* happen at T_{next} or bring about the property specified by *Fluent* (with the appropriate *Parameters*) at T_{next}, respectively, with T_{next} satisfying the temporal constraints

TCs. Example instances of the first rule above are given in section 4.

Obligations will be interpreted differently by different agents, depending on their personalities. For example, a "responsible" agent will try to fulfill all its obligations, if possible. We can specify this with reactive rules of the form:

$$holds\_at(responsible(Actor), T) \wedge$$
$$obliged(act(Act, Actor), T, TCs) \Rightarrow$$
$$happens(Act(Actor), T) \wedge TCs$$

$$holds\_at(responsible(Actor), T) \wedge$$
$$obliged(fluent(Fluent, Actor), T, TCs) \Rightarrow$$
$$holds\_at(Fluent(Actor), T) \wedge TCs$$

referred to as $(R1)$ and $(R2)$, respectively.

3.4 Prohibitions

Prohibitions can be specified in a similar way to obligations. We use atoms of the form

$prohibited(act(Act, Actor), T, TCs)$
$prohibited(fluent(Fluent, Actor), T, TCs)$

to indicate that $Actor$ is prohibited from performing Act (with the appropriate *Parameters*) at time T within the time constraints TCs, and $Actor$ is prohibited from bringing about $Fluent$ (with the appropriate *Parameters*) to hold at time T within the time constraints TCs, respectively.

Similarly to obligations, prohibitions for individual agents are generated by means of reactive rules in KB_{react} in the knowledge base of the agents. These reactive rules take the general forms:

$$holds\_at(role(Actor, Role), T_{now}) \wedge$$
$$Conditions(Actor, T_{now}) \Rightarrow$$
$$prohibited(act(Act, Actor), T_{next}, TCs))$$

$$holds\_at(role(Actor, Role), T_{now}) \wedge$$
$$Conditions(Actor, T_{now}) \Rightarrow$$
$$prohibited(fluent(Fluent, Actor), T_{next}, TCs))$$

A concrete example of the first rule is

$$holds\_at(role(X, t\_w(W, F, T), T') \wedge$$
$$F \leq T' \leq T \Rightarrow$$
$$prohibited(act(leave(W), X), T'', T'' = T')$$

This rule says that a traffic warden is prohibited to leave his allocated area while he is on duty.
The following rules added to KB_{plan} of the agents then specify the behaviour of responsible agents in the light of prohibitions:

$$holds\_at(responsible(Actor), T) \wedge$$
$$prohibited(act(Act, Actor), T, TCs) \wedge$$
$$happens(Act(Actor), T) \wedge TCs \Rightarrow false$$

$$holds\_at(responsible(Actor), T) \wedge$$
$$prohibited(fluent(Fluent, Actor), T, TCs) \wedge$$
$$holds\_at(Fluent(Actor), T) \wedge TCs \Rightarrow false$$

These rules indicate that a responsible agent does not perform prohibited acts and does not bring about prohibited fluents to hold.

4 An Example

In this section we exemplify the use of the proposed simple formalisation of normative concepts for "responsible" KGP agents as an executable specification, to which the behaviour of the agents is guaranteed to conform.

Suppose we have two agents, ag_1 and ag_2, with ag_1 a responsible owner of car $W129FGC$ and ag_2 a responsible traffic warden in the $chelsea$ area from time 9 to time 17. They both hold, in addition to rules $(R1)$ and $(R2)$ above, the following reactive rules in their KB_{react}:

$$holds\_at(role(X, t\_w(W, F, T), T') \wedge$$
$$observed(parked(C, W), T') \wedge$$
$$F < T' < T \Rightarrow$$
$$obliged(act(issue\_fine(C), X), T'', T'' = T' + 1))$$

$$holds\_at(role(X, owner(C)), T) \wedge$$
$$observed(issued\_fine(C), T) \Rightarrow$$
$$obliged(act(pay\_fine(C), X), T', T' = T + 5))$$

referred to as $(R3)$ and $(R4)$, respectively.

Assume now that, starting at time 10, agent ag_1 performs three transitions in sequence: Passive Observation Introduction (POI), Reactivity (RE), and

Action Execution (AE). Suppose through the POI, at time 10, agent ag_1 observes that car $W129FGC$ is parked in the $chelsea$ area. POI records this observation in the KB of ag_1 in the form of the atom

$$observed(parked(''W129FGC'', chelsea), 10)$$

Then RE allows ag_1 to generate for itself the action of issuing a fine to the car at time 11. This is done first by the observation triggering reactive rule $(R3)$ resulting in the derivation of

$$\begin{aligned}&obliged(act(issue\_fine(''W129FGC''), ag_1),\\&\qquad T,\\&\qquad T = 10 + 1)\end{aligned}$$

which, in turn, triggers reactive rule $(R1)$, resulting in the derivation of

$$\begin{aligned}&happens(issue\_fine(ag_1,'' W129FGC''), T) \wedge\\&T = 10 + 1\end{aligned}$$

The RE transition adds the action to the $Plan$ of agent ag_1. Then the AE transition selects and executes the action by adding a record of its execution to the KB of ag_1.

Now assume that, starting from time 11, ag_2 executes the POI and RE transitions respectively. POI allows ag_2 to observe that it has been issued a fine:

$$observed(issued\_fine(''W129FGC''), 11)$$

RE then allows ag_2 to generate first an obligation for itself to pay the fine:

$$\begin{aligned}&obliged(act(pay\_fine(''W129FGC''), ag_2),\\&\qquad T,\\&\qquad T = 11 + 5)\end{aligned}$$

and then an action

$$\begin{aligned}&happens(pay\_fine(ag_2, W129FGC), T) \wedge\\&T = 11 + 5\end{aligned}$$

which will be added to ag_2's $Plan$. If ag_2 then applies the Plan Introduction transition PI next, this may lead to further changes in its $Plan$ and $Goals$. For example in KB_{plan} of ag_2 the preconditions of pay_fine may be specified as follows:

$$precondition(pay\_fine(X, Y), have\_money(X))$$
$$precondition(pay\_fine(X, Y), at(X, police\_station))$$

In this case two news subgoals may be added to ag_2's $Goals$, namely to have money and be at the police station at time 16. These would then be further planned for using KB_{plan} and transition PI.

5 Related Work

Computational logic approaches to specify the rules of the interaction amongst social agents is an active subject of research. The existing society model (Alberti et al., 2004a) of PROSOCS, for example, is concerned with the specification of public protocols for communicating agents. This model provides a general language for specifying agent-protocols using the notion of expectations as well as a tool that supports verifiction of compliance for these protocols. Instead, our work presented here seeks to complement what we already have in PROSOCS, so that agents can reason and plan with private obligations, so that we distribute and communicate normative concepts such as roles and obligations to different agents, as it normally happens in a human society. We believe to be able to reuse the mapping between expectations and normative concepts discussed in (Alberti et al., 2004b).

The work described in (Artikis et al., 2002) is also based on computational logic and consists of a theoretical framework for providing executable specifications of particural kinds of multi-agent systems, called open computational societies. Three key components are introduced: a social state, social roles, and social constraints. The specification of these concepts is based on and motivated by the study of legal and social systems, and a representation of deontic concepts in a version of the event calculus is presented.

Our framework differs from that of (Artikis et al., 2002) in that we focus on how to reason about social constraints and roles within a single agent using an abductive interpretation of the event calculus. In other words, we do not take a "bird's eye view" of the interaction between agent but an "agent's view", trying to interpret social constraints from the standpoint of a single agent (based on the specific agent architecture of the KGP model) and further showing how such constraints can be used during agent deliberation.

The IMPACT system (Arisha et al., 1999) incorporates obligations and other deontic notions such as permission and prohibition. In common with us they use abductive logic programs as the representation language. But IMPACT incorporates a more extensive theory of deontic concepts, as well as rules for allowing the utilisation of legacy systems. The KGP model, and its implementation in PROSOCS have different aims compared to IMPACT. We aim at building agents that can plan partially, interleave planning, acting and observations, and adaptability of agents incorporating flexible cycle theories that mimic an agent's personality.

The BOID architecture presented in (Broersen et al., 2001) extends the well known BDI model (Rao and Georgeff, 1995) with obligations, thus giving rise to four main components in representing an agent: beliefs, obligations, intentions and desires. The idea of BOID is to find ways of resolving *internal* conficts within each of the four components and across two or more of the components. In order to do so they define agent types including some well known types in agent theories such as realistic, selfish, social and simple minded agents. The agent types differ in that they give different priorities to the rules for each of the four components. For instance, the simple minded agent gives higher priority to intentions, compared to desires and obligations, whereas a social agent gives higher priority to obligations than desires. The use of priorities is with propositional logic formulae to specify the four components and the agent types.

What we have in common with BOID is that we want to extend our model with the addition of obligations. The existing KGP model already resolves some of the conflicts that they address. For example, if there is a conflict between a belief and a prior intention, which means that an intended action can no longer be executed due to the changes in the environment, the KGP agent will notice this and will give higher priority to the belief than the prior intention, allowing the agent in effect to rectract the intended action and, time permitting, to replan for its goals. The KGP model also includes a notion of priority used in the goal decision capability and the cycle theory that controls the behaviour of the agent.

One of the aims of the KGP model and the extension discussed in this paper, and one of the differences with the work of (Broersen et al., 2001), is to allow the interleaving of concluding obligations, planning to achieve them and other goals and recording and utilising observations. This makes our work closer to the more recent work on BOID, see for instance (Dastani and van der Torre, 2004), by providing an executable specification of the obligations in an abductive logic programming setting. However, our work is not about resolving conflicts between obligations and intentions as that of (Dastani and van der Torre, 2004); we do however plan to study conficts of this kind in future work.

6 Concluding Remarks

We have illustrated how to extend the logical model of agency, known as the KGP model, to support agents with normative concepts. By using obligations and prohibitions as an example, the proposed framework has shown how to specify an agent that can reason using normative concepts and combine them in order to plan and decide which action the agent should perform next.

Unlike approaches that are based on a monolithic tool for checking social interactions, the advantage of this work is that obligations and prohibitions can be interpreted within a social agent and communicated to other agents using the KGP model, thus making the extended model suitable for building multi-agent systems applications whose organisation is based on artificial and open societies of agents. This is an issue for future research.

Future work also involves incorporating permissions, rewards and sanctions in the current model and implementing the extended model in the PROSOCS platform to experiment with concrete applications. Finally, future work involves importing into the KGP model more sophisticated normative theories.

Acknowledgments

The authors would like to thank the anonymous referees for their comments and suggestions on an earlier version of this paper. This work was partially funded by the IST programme of the European Commission, Future and Emerging Technologies under the IST-2001-32530 SOCS project, within the Global Computing proactive initiative.

References

Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Compliance verification of agent interaction: a logic-based tool. In Robert Trappl, editor, *Proceedings of the 17th European Meeting on Cybernetics and Systems Research, Vol. II, Symposium "From Agent Theory to Agent Implementation" (AT2AI-4)*, pages 570–575, Vienna, Austria, April 13-16 2004a. Austrian Society for Cybernetic Studies.

Marco Alberti, Evelina Lamma, Marco Gavanelli, Paola Mello, Giovanni Sartor, and Paolo Torroni.

Towards a mapping of deontic logic onto an abductive framework. In *Workshop on Agents and Constraints (Agenti & Vingoli), AI*IA*, 2004b.

K. A. Arisha, F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus. IMPACT: a Platform for Collaborating Agents. *IEEE Intelligent Systems*, 14(2):64–72, March/April 1999.

A. Artikis and J. Pitt. A formal model of open agent societies. In J. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of Conference on Autonomous Agents (AA)*, pages 192–193. ACM Press, 2001.

A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In C. Castelfranchi and L. Johnson, editors, *Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1053–1062. ACM Press, 2002.

A. Bracciali, N. Demetriou, U. Endriss, A. Kakas, W. Lu, P. Mancarella, F. Sadri, K. Stathis, G. Terreni, and F. Toni. The KGP Model for Global Computing: Computational Model and Prototype implementation. In *Global Computing*, LNCS. Springer-Verlag, 2004. To appear.

Jan Broersen, Mehdi Dastani, Joris Hulstijn, Zisheng Huang, and Leendert der van Torre. The BOID architecture: conflicts between beliefs, obligations, intentions and desires. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 9–16, Montreal, Canada, 2001. ACM Press. URL `citeseer.ist.psu.edu/broersen01boid.html`.

C. Carabelea, O. Boissier, and C. Castelfranchi. Using social power to enable agents to reason about being part of a group. In *Pre-proceedings of ESAW'04*, Toulouse, October 2004.

Cristiano Castelfranchi, Frank Dignum, Catholijn M. Jonker, and Jan Treur. Deliberative normative agents: Principles and architecture. In *Agent Theories, Architectures, and Languages*, pages 364–378, 1999. URL `citeseer.lcs.mit.edu/castelfranchi99deliberative.html`.

M. Dastani and L. van der Torre. Programming boid agents: a deliberation language for conflicts between mental attitudes and plans. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'04)*, 2004.

Frank Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999. URL `citeseer.ist.psu.edu/article/dignum99autonomous.ht`

Michael N. Huhns and Munindar P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, San Francisco, CA, USA, 1998.

A.J.I. Jones and M.J. Sergot. On the characterisation of law and computer systems: the normative systems perspective, 1993. URL `citeseer.ist.psu.edu/jones93characterisation.html`. Deontic Logic in Computer Science: Normative System Specification. John Wiley and Sons, Chicester (1993) 275–307.

A.J.I. Jones and M.J. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, June 1996.

A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. Declarative agent control. In Leite J. and Torroni P., editors, *Proceedings CLIMA'04, 5th International Workshop on Computational Logic in Multi-Agent Systems*, Lisbon, Portugal, Sep. 2004a.

A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *European Conference on Artificial Intelligence (ECAI04)*, pages 33–39, 2004b.

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

Julian A. Padget, editor. *Collaboration between human and Artificial Societies: Coordination and Agent-based Distributed Computing*. Springer, LNAI 1624, 2001.

A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995. URL `citeseer.ist.psu.edu/rao95bdi.html`.

K. Stathis, A. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: a platform for programming software agents in computational logic. In J. Müller and P. Petta, editors, *Proceedings of the Fourth International Symposium "From Agent Theory to Agent Implementation"*, Vienna, Austria, April 13-16 2004.

F. Toni and K. Stathis. Access-as-you-need: a computational logic framework for flexible resource access in artificial societies. In *Proceedings of the Third International Workshop on Engineering Societies in the Agents World (ESAW'02)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2002.