The KGP Model of Agency

Antonis Kakas [1] and **Paolo Mancarella** [2] and **Fariba Sadri** [3] and **Kostas Stathis** [4] and **Francesca Toni** [3]

Abstract. This paper presents a new model of agency, called the KGP (**K**nowledge, **G**oals and **P**lan) model. This draws from the classic BDI model and proposes a hierarchical agent architecture with a highly modular structure that synthesises various reasoning and sensing capabilities of the agent in an open and dynamic environment. The novel features of the model include: its innovative use of Computational Logic (CL) in a way that facilitates both the formal analysis of the model and its computational realisability directly from the high-level specification of the agents (a first prototype for the development of KGP agents exists, based upon a correct computational counterpart of the model), the modular separation of concerns and flexibility afforded by the model in designing heterogeneous agents and in developing independently the various components of an agent, and the declarative agent control provided through a context-sensitive cycle CL theory component that regulates the agent's operational behaviour, according to the current circumstances of operation, thus breaking away from the conventional one-size-fits-all control of operation.

1 INTRODUCTION

Agents are situated autonomous entities that are expected to plan and act to achieve their goals while they are alert and responsive to the changes in their environments (see e.g. [24]). In this paper we present a new model of agency called the KGP (**K**nowledge, **G**oals and **P**lan) model. This provides a highly modular and hierarchical specification of agents equipped with a variety of advanced reasoning features to allow intelligent decision making and behaviour. KGP agents are particularly suited to open, dynamic environments where they have to adapt to changes in their environment and they have to function in circumstances where they have incomplete information.

KGP is motivated, on the one hand, by the existing gap between modal logic specifications [20] of BDI agents [3] and their implementation (see issues raised by Rao in [19]) and, on the other, to make available and extend many useful computational logic (CL) [12] tools and techniques whose synthesis can produce executable specifications of agents. For this purpose, the model synthesises Abductive Logic Programming (ALP) [13] and Logic Programming with Priorities (LPP) [18], both extended to deal with constraint solving as in Constraint Logic Programming (CLP) [10].

CL is used in KGP to specify the individual *state* of the agent, its reasoning *capabilities*, state *transitions*, and its *control*. Using these components, an agent maintains a view of the environment, decides what its goals should be, depending on the current circumstances, plans (incrementally) for these chosen goals and interleaves this with incremental execution of the plan, reacts to information received from the environment or communication received from other agents, re-evaluates previous decisions in the light of the new information and adapts as necessary by changing or augmenting its goals and plan. The control component of this complex behaviour is regulated by a *cycle theory*, which also allows us to design agents with a wide range of heterogeneous behaviour suitable for different practical applications.

As a consequence of the CL-based approach, the resulting declarative model comes coupled with a fully specified and correct computational model. We believe that the grounding of the declarative and the computational models in CL also highly facilitates formal verification of the different levels of the model and proof of various properties. For the purposes of experimentation, a first prototype implementation has already been completed successfully using SICStus Prolog, Java, and JXTA [22].

The KGP model was developed as part of the EU (IST: FET - Global Computing) research project, SOCS (Societies of Computees) http://lia.deis.unibo.it/research/projects/socs. Details of the declarative and computational models together with examples can be found in the deliverables of this project [11, 14, 1].

The rest of the paper is structured as follows. We describe the organisation of the internal state of a KGP agent in section 2 and the way reasoning capabilities access this state in section 3. In section 4 we summarise how transitions use capabilities to change the state of the agent and in section 5 we explain how the cycle theory uses transitions to control the behaviour of the agent. Finally, in section 6 we compare the KGP model with existing work and conclude in section 7.

2 OVERVIEW OF THE KGP MODEL

As shown in Fig.1, the basis of the hierarchical KGP model is the *knowledge* of the agent. This is accessed by a modular collection of *capabilities* that enable the agent to plan or react, decide new goals, reason temporally and sense the environment in order to check whether goals or (for cautious agents) the preconditions of actions in plans are satisfied. Capabilities are utilised in a collection of *transitions* that describe how the internal state of the agent changes. Changes of state may also occur by the agent observing either actively, in order to test whether something holds in the environment, or passively, as a result of being situated in a specific environment. Changes may also result because of the agent choosing to execute actions, sense the environment, introduce new goals or plans, revise these goals and plans, or simply react to changes in the environment. The transitions are integrated within dynamic and flexible *cycle the-*

[1] Department of Computer Science, University of Cyprus, Nicosia CY-1678, Cyprus. email: antonis@cs.ucy.cy

[2] Dipartimento di Informatica, Universita di Pisa, Pisa 56127, Italy. email: paolo@di.unipi.it

[3] Department of Computing, Imperial College, London SW7 2AZ, UK. email: {fs,ft}@doc.ic.ac.uk

[4] Department of Computing, City University, London EC1V 0HB, UK. email: kostas@soi.city.ac.uk

ories that specify declaratively how the transitions are sequenced depending on the environment and the required behaviour profile of the agent (e.g. whether we want them to be cautious or careful).

The internal state of a KGP agent is a triple $\langle KB, Goals, Plan\rangle$. The *Knowledge Base* (KB) describes the knowledge of the agent of itself and its environment. It consists of separate modules supporting the different reasoning capabilities, for example, KB_{plan} contains knowledge that enables the agent to plan and KB_{GD} contains knowledge that enables the agent to decide which goals to adopt next. One part of the KB, called KB_0, holds the (dynamic) knowledge of the agent about its external world, including what the agent has observed, the actions it has executed, and communications it has received from other agents. For example, we may express what the agent has observed from the environment with assertions of the form $observed(l, t)$, where l is a fluent literal and t is a time constant. This fact is added when the property l has been observed to hold at time t. We assume that KB_0 is the only part of the KB that changes over time.

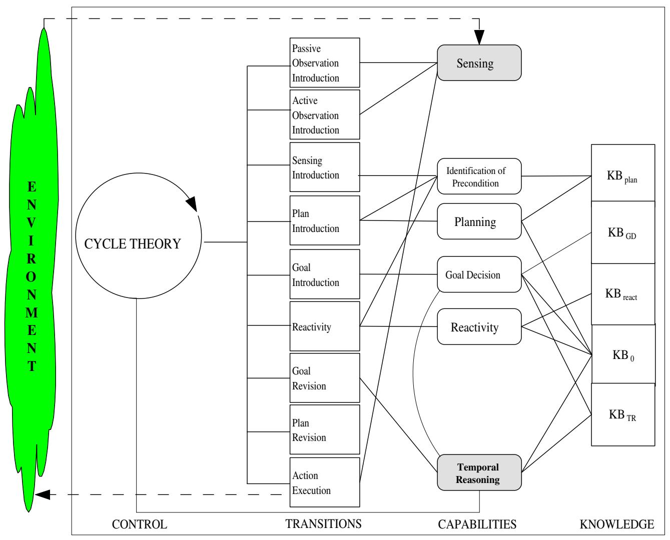

Figure 1. The conceptual organisation of a KGP agent.

The *Goals* of an agent is a set of properties that the agent has decided that it wants (desires) to achieve each at a given time, within certain temporal intervals, specified by temporal constraints. $Goals$ are split into two types: *mental goals*, that can be planned for by the agent, and *sensing goals*, that can only be sensed (to find out from the environment whether they hold or not). As an example, consider the goal $holds\_at(have\_fuel, t) \wedge t < 10$, meaning that the agent will have to satisfy the property $have\_fuel$ by time 10.

The *Plan* of an agent is a set of partially ordered (atomic) actions by means of which the agent plans (intends) to satisfy its goals. Actions are split into three types: *physical actions*, that the agent can execute to effect a change in the external world, *communicative actions*, used by the agent to exchange information with other agents, e.g. to make a request, and *sensing actions*, for getting information from the environment. Actions are to be executed at given times, within temporal intervals specified by temporal constraints (as for goals). As an example, consider the action $happens(fill\_up, t') \wedge t-2 < t' < t$, meaning that the agent will have to execute the $fill\_up$ action in the temporal interval $(t-2, t)$. Actions that are to be added to the $Plan$ are identified by the capabilities of the agent, which we describe in the next section. The associated time for goals and actions is either ground or assumed to be existentially quantified over the whole state.

3 CAPABILITIES

As shown in Fig. 1, the KB of a KGP agent is used by a set of *reasoning capabilities* that are specified in ALP and LPP. These capabilities enable the agent to perform different reasoning tasks while adapting to its environment.

- A *Planning* capability uses KB_{Plan} and KB_0 to generate *partial plans*, i.e. sets of actions and (sub)goals, for a given set of goals. This allows the agent to be *adaptable* to changes in the environment without wasting planning effort.
- An *Identification of Preconditions* capability uses KB_{Plan} to identify the preconditions for the successful execution of given actions. This capability is needed for "cautious" agents which might want to sense the environment prior to execution of actions to make sure that the execution will be successful. In addition, by means of this capability an agent may realise that the required preconditions of some actions will never be satisfied, and thus those actions may be dropped from the current $Plan$, thus allowing for adaptability of agents.
- A *Goal Decision* capability uses KB_{GD} and KB_0 to determine the top-level goals that the agent prefers to achieve under the current circumstances according to its preference policy in KB_{GD}.
- A *Temporal Reasoning* capability (which is also used within some other capabilities and components of the model) uses KB_{TR} and enables the agent to reason from observations stored in KB_0 and make predictions about properties on the basis of these observations. It plays a fundamental role in rendering the agent *adaptable* to changes in its environment by dealing appropriately with *partial information* which evolves over time.
- A *Reactivity* capability allows the agent to react to changes in the external environment. Reactivity uses KB_{react} and KB_0 to identify which new actions and new goals should be added to the current state of the agent in the light of the new observations perceived in the environment.

To exemplify how a reasoning capability is specified, let us consider Reactivity in more detail. The KB_{react} part of the knowledge base of an agent contains an *abductive event calculus* [15] theory T_{ec} (which we omit here due to space limitations) along with some conditions-action-like rules, called *reactive rules* of the form:

$$Conditions \rightarrow Reaction$$

where $Reaction$ is either a goal or an action and $Conditions$ is a non-empty conjunction of conditions on the current state of the agent. $Conditions$ are typically checked in KB_0 and the agent's Goals and Plan via T_{ec}, and act as *triggers* which (may) fire the applicability of the reactive rule. For example, the reactive rule:

$$holds\_at(drive\_home, t) \wedge observed(low\_fuel, t') \wedge t' < t \rightarrow$$
$$assume\_holds(have\_fuel, t'') \wedge t' < t'' < t.$$

states that if one of the agent's current goals is to drive home at some time point t, and the agent has observed that there is low fuel at an earlier time point t', then the *new* goal of having fuel at some time point between t' and t has to be added to the current set of goals. Here the goals of the agent are formulated in the abductive event calculus style: an atom of the form $holds\_at(p, t)$ means that the proposition p holds at some time point t. The predicate $assume\_holds$ is the abductive variant of $holds\_at$. Whereas atoms expressed by means of $holds\_at$ can be reasoned with through the event calculus background theory, atoms of the form $assume\_holds$ can only be abductively assumed during the reasoning process. For Reactivity, these abductive assumptions represent the new goals the agent should add to its current state in order to satisfy its reactive rules.

As a further example, consider the following reactive rule:
$holds\_at(have\_fuel, t) \wedge observed(out\_of\_money, t') \wedge t' < t \rightarrow$
$assume\_happens(withdraw\_money, t'') \wedge t' < t'' < t.$
Such a rule forces the agent to execute the action of withdrawing money if the goal of having fuel is in the current set of goals and the agent has observed that it has run out of money. As for goals, actions are represented here in the abductive event calculus style: $happens(a, t)$ means that the action represented by a has to be (or has been) executed at time t. The predicate $assume\_happens$ is the abductive variant of $happens$, representing actions that need to be added to the current $Plan$ to fulfill the reactive rules.

The T_{ec} background theory describes the causal relationships between events (actions) and fluents (goals) in the event calculus style. KB_{react} is then $T_{ec} \cup RR$, where RR is a given set of reactive rules as described above. Let S^* denote the KB_0, Goals, and Plan components of an agent's state S. Then the Reactivity capability yields a set Gs of goals and a set As of actions which are abductively entailed by $KB_{react} \cup S^*$, along the lines sketched above. Thus, more formally, we will write:
$KB_{react} \cup S^* \models^{\tau}_{react} Gs, As$
meaning that Gs, As is the result of the Reactivity capability at time point τ in the state S. Note that the Reactivity capability ensures that the new goals Gs and new actions As are consistent with the existing $Goals$ and $Plan$ and that they are indeed achievable at some time in the future of τ .

In addition to the reasoning capabilities and their example specification above, the agent is assumed to be equipped at the implementation level with a *Sensing* capability that links the agent with the environment, by allowing it to observe properties holding in the environment and be informed about actions being executed by other agents, all of which are stored in KB_0.

4 TRANSITIONS

As shown in Fig.1, the capabilities of a KGP agent described in the previous section are used in *transition rules* whose application changes the agent's internal state, as outlined below.

- *Passive Observation Introduction* (POI) changes KB_0 by introducing unsolicited information from the environment or communications from other agents. POI calls the Sensing capability.
- *Active Observation Introduction* (AOI) changes KB_0 by recording the outcome of sensing actions for actively sought properties. AOI calls the Sensing capability.
- *Sensing Introduction* (SI) transition adds to the current *Plan* new sensing actions for sensing the preconditions of actions already in $Plan$, and uses the Sensing capability.
- *Plan Introduction* (PI) changes part of the $Goals$ and $Plan$ 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 action A in the set computed by Planning, with the set of preconditions for the successful execution of A. However, this does not necessarily mean that such preconditions will be checked at the time of the execution of the actions.
- *Goal Introduction* (GI) replaces the $Goals$ of a state with goals of highest priority that the Goal Decision capability generates.
- *Reactivity* (RE) updates the current state of the agent by adding the goals and actions returned by the Reactivity capability. As with PI, this transition uses the Identification of Preconditions capability to equip each action A in the set computed by Reactivity, with the set of preconditions for the successful execution of A.
- *Goal Revision* (GR) revises $Goals$, e.g. by dropping goals that have already been achieved or that have run out of time, by using the Temporal Reasoning capability and by checking the temporal constraints of the goals.
- *Plan Revision* (PR) revises $Plan$, e.g. by dropping actions that have already been executed successfully or that have run out of time, by checking the temporal constraints of the actions.
- *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. Calls the Sensing capability for the execution of sensing actions.

To exemplify how a transition rule is specified, we show here how the Reactivity transition rule is specified, by changing the goals and plan of the agent at a time τ as follows:

$$\textbf{(RE)} \qquad \frac{\langle KB, Goals, Plan \rangle}{\langle KB, Goals', Plan' \rangle}\, \tau$$

where, given $KB_{react} \models^{\tau}_{react} Gs, As$, then $Goals' = Goals \cup Gs$ and $Plan' = Plan \cup As$.

5 CYCLE THEORY

The operation of a KGP agent is controlled by a cycle theory, whose role is not to provide fixed control on the operation of the agent but to regulate it in a way that results in a desired pattern of behaviour. Cycle theories are written in the framework of LPP, for which we adopt the concrete framework of LPwNF [7] suitably extended to deal with conditional, dynamic priorities [6]. Other frameworks for LPP (e.g. [18]) or for the declarative specification of preference policies, (e.g. [5]), can be used.

The LPwNF framework, as any other LPP framework, is equipped with a notion of entailment, that we refer to as $\models_{pr}$. Intuitively, given a theory $\mathcal{T}$, and a literal L, $\mathcal{T} \models_{pr} L$ means that L is entailed by a sub-theory of $\mathcal{T}$ which is "preferred", according to the strength of the rules given by the priorities specified in $\mathcal{T}$, over any sub-theory of $\mathcal{T}$ that derives a conclusion incompatible with L. In this paper, we will assume for simplicity that $\models_{pr}$ always entails one and only one conclusion, namely there exists exactly one L such that $\mathcal{T} \models_{pr} L$.

In presenting the cycle theory, we will assume that transitions are represented as atoms, $T_i(S, X, S', \tau)$, where S is the state of the agent before the transition is applied and S' the state after the transition is applied, X is the input (possibly empty) taken by the transition, and τ is the time of application of the transition. The subscript i is the name of the transition rule, i.e. it is an element from the set $I = \{POI, AOI, SI, PI, GI, RE, GR, PR, AE\}$, applied in this transition. We assume the existence of a *clock* (possibly external to and shared by the agents) which marks the passing of time. In the sequel, for simplicity, we may drop some of the parameters of the transitions.

Given a cycle theory, T_{cycle}, the basic *unitary operational step* of the agent is specified as follows:

$$T_{cycle} \cup \{T_m(S_{i-1}, X_i, S_i, \tau_i)\} \models_{pr} T_n(S_i, X_{i+1}, S_{i+1}, \tau_{i+1})$$

where $T_m(S_{i-1}, X_i, S_i, \tau_i)$ is the last transition, executed at time τ_i and $T_n(S_i, X_{i+1}, S_{i+1}, \tau_{i+1})$ is the one to follow.

Concretely, a cycle theory consists of three components:

- A *basic* part that specifies the allowed unitary cycle-steps from one transition to the next.
- An *interrupt* part that specifies the cycle-steps that can follow a POI, i.e. an interrupt with new information. These are viewed as (possible) re-initialisation steps for the cycle operation.
- A *behaviour* part that specifies priority rules on the alternatives given in the basic and interrupt parts, and thus specifies the special characteristics of the operation of the agent.

The basic part of any cycle theory consists of rules of the form:

$$r_{i|k}(S', X) : T_k(S', X) \leftarrow T_i(S, S'), C_{i|k}(S', X).$$

where T_i and T_k are any two transitions different from POI, i.e. $i, k \neq POI$. Here $r_{i|k}(S', X)$ is a term that names (the instances of) this rule. Such a rule specifies that transition T_k might follow a transition T_i. We will assume that the step to the next transition depends only on the current transition and not on the longer history of the previous transitions. Note that this does not mean that information from the past operations of the agent is not used as such information is recorded and assimilated in the state of the agent.

The conditions $C_{i|k}$ in such a cycle-step rule are called *Enabling Conditions* as they determine when a cycle-step from T_i to T_k is allowed or enabled. In particular, they determine the input X, if any is required, of the ensuing transition T_k. Such input will be determined by calls to appropriate *Selection Functions*, when required. For example, the following cycle-step rule:

$$r_{PI|AE}(S', As) : T_{AE}(S', As) \leftarrow T_{PI}(S, S'), C_{PI|AE}(S', As).$$

expresses the possibility that a PI transition can be followed by an AE transition. The Enabling Conditions $C_{PI|AE}(S', As)$ determine the set of actions As that are to be executed by the ensuing AE transition. These are given by a *Core Action Selection* function that selects only actions that can be executed, e.g. actions that have not been timed out and whose preconditions are not known to be currently false.

The interrupt component of the cycle theory is analogous, in syntax, to the basic component. However, each rule in the interrupt theory specifies what might follow a POI transition, which acts as an interrupt. Concretely, an interrupt cycle-step rule is of the form:

$$r_{POI|k}(S', X) : T_k(S', X) \leftarrow T_{POI}(S, S'), C_{POI|k}(S', X).$$

where $k \in I$ and $k \neq POI$. For example, after a Passive Observation Introduction we may want to apply the Reactivity (RE) transition so that we can adapt the existing $Plan$ to the new information that the observation gives us. This is achieved by the simple interrupt cycle-step rule:

$$r_{POI|RE}(S', X) : T_{RE}(S', \{\}) \leftarrow T_{POI}(S, S').$$

The behaviour part of the cycle theory consists of priority relations that encode locally the relative strength of the rules in the other components of the cycle theory. These then are used to determine, amongst all the enabled cycle-steps, which one is preferred under the current circumstances. They have the form:

$$R^i_{k|l} : r_{i|k}(S, X_k) > r_{i|l}(S, X_l) \leftarrow BC^i_{k|l}(S, X_k, X_l)$$

where $BC^i_{k|l}$ are called *Behaviour Conditions*. These are conditions, e.g. *Heuristic Selection Functions*, under which the cycle-step to transition T_k with input X_k is preferred over that of T_l with input X_l.

Through the behaviour part of the cycle theory we can encode different patterns of operation thus allowing heterogeneity of agents. For example, by giving priority to the cycle-step rule of GR over any other cycle-step rule, after a POI or AOI transition, we have a *careful* pattern of behaviour whereby the agent examines and revises its current commitments in the light of the new information obtained. Similarly, by giving priority to cycle-steps of AOI on the effect of an action after an AE transition, we have a *cautious* pattern of behaviour where the agent attempts to get explicit confirmation of the result of the execution of its actions.

A simple conventional or normal pattern of behaviour can be one whereby an agent prefers to follow the pattern: introduce goals (GI), plan for them (PI), execute the actions of the plan (AE), revise the state (GR, PR), return to continue planning (PI) until all goals are dealt with (successfully completed or revised away) and then return to introduce new goals (GI). It is easy to see that we can get *fixed cycles* of operation as a special case of cycle theories where only the specified cycle-steps are enabled in the basic part of the cycle theory and where the behaviour part is empty. Fixed cycles of operation correspond to unconditional priority rules in the behaviour part of the cycle theory. The conditional rules in the behaviour part thus generalise the fixed cycle of operation of agents allowing a KGP agent to be adaptable to the changing conditions of its environment as it operates.

6 RELATED WORK

Many proposals for models and architectures of individual agents exist. Some of these proposals are based on logic programming, for example IMPACT [2], Minerva [16], GOLOG [17] , and IndiGolog [8]. Other proposals are based on modal or first order logic or are not logic based, for example the BDI model [3, 20], Agent0 [21], AgentSpeak [19] and its variants, 3APL [9], and DESIRE [4].

At a high level of comparison there are similarities in the objectives of these models and the KGP model, in that they all aim at specifying knowledge-rich agents with certain desirable behaviours. There are also some similarities in the finer details of the KGP model and some of the above related work, as well as differences.

A novel feature of the KGP model is its declarative and context-sensitive specification of the control component of the agent. The cycle theory of the KGP agent determines, at run time, depending on the circumstances and the individual profile of the agents, what their next step should be. The cycle theory is sensitive to both solicited and unsolicited information that the agent receives from its environment.

Another distinguishing feature of the KGP model, in comparison with other models, including those based on logic programming, is its modular integration within a single framework of ALP, CLP, and preference reasoning based on LPP, in order to support a diverse collection of capabilities. Each one of these is specified declaratively and equipped with its own provably correct computational counterpart.

There is an obvious similarity between the KGP and the BDI models [3] given by the correspondence between KGP's Knowledge, Goals and Plan and BDI's Beliefs, Desires and Intentions, respectively. However, the BDI model is based on modal logic and the gap between its specification and its practical realisation is much wider than with KGP. The same difference exists between the KGP model and Agent0 and its later refinement PLACA [23]. AgentSpeak(L) [19] attempts to narrow the gap between the specification and executable model of BDI and in that it shares one of the objectives of the KGP. Two other differences between the KGP and Agent0 and

PLACA are the explicit links that exist in the KGP model amongst the goals and between the goals and plans, and the richer theories in the KGP that specify priorities amongst potential goals which are not restricted to temporal orderings. These explicit links are exploited when revising goals and plans, via the Goal Revision and Plan Revision transitions, in the light of new information or because of the passage of time.

MINERVA [16] has two main similarities with the KGP; it exploits computational logic and it gives both declarative and operational semantics to its agents. It has a number of differences, e.g. MINERVA relies on Multidimensional Dynamic Logic Programming [16] and uses explicit rules for updating its knowledge bases. IndiGolog [8] is a high-level programming language for robots and intelligent agents that supports, like KGP, on-line planning, sensing and plan execution in dynamic and incompletely known environments. It is a member of the Golog family of languages [17] that use a Situation Calculus theory of action to perform the reasoning required in executing the program. Instead in the KGP model we rely on ALP and LPP combined with an Event Calculus approach to program an agent. Moreover, in IndiGolog goals cannot be decided dynamically, whereas in the KGP model they change dynamically via the Goal Decision capability.

There are features in some other approaches that are absent in the KGP model. BDI and more so the IMPACT system [2] allow agents to have in their knowledge bases representations of the knowledge of other agents. These systems allow the agents both some degree of introspection and ability to reason about other agents' beliefs and reasoning. IMPACT also allows the incorporation of legacy systems, and has a richer knowledge base language allowing deontic concepts (obligation, permission, etc) and probabilities. 3APL, based on a combination of imperative and logic programming languages, includes an optimisation component absent from the KGP. This component in 3APL includes rules that identify if the agent is pursuing a suboptimal plan, and help the agent find a better one.

7 CONCLUSIONS

KGP is a logical model of agency based on a hierarchical, modular, and extensible agent architecture, which is characterised by the innovative use of CL to facilitate the formal analysis and computational realisability of an agent from a high-level specification of its knowledge, goals, and plans. The model identifies a set of reasoning capabilities, a set of transitions, and a context-sensitive cycle theory that regulates an agent's operational behaviour, according to the current circumstances, and thus breaks away from traditional approaches of one-size-fits-all control of operation. The model is also equipped with a computational counterpart, based upon an abductive proof-procedure (for the ALP components) and an argumentation-proof procedure (for the LPP components). A prototype implementation platform for the development of KGP agents already exists.

Future work includes extending the model to incorporate (i) knowledge revision (e.g. by Inductive Logic Programming), (ii) introspective reasoning and reasoning about the beliefs of other agents, (iii) dealing with society expectations and deontic concepts (e.g. as in IMPACT), (iv) further experimenting with the model via its implementation, (v) verifying formally a catalogue of behaviour properties of agents equipped with different cycle theories.

ACKNOWLEDGEMENTS

This work was supported by the European Commission FET Global Computing Initiative, within the SOCS project (IST-2001-32530).

REFERENCES

[1] M. Alberti, A. Bracciali, F. Chesani, N. Demetriou, U. Endriss, M. Gavanelli, A.C. Kakas, W. Lu, K. Stathis, and Paolo Torroni, 'SOCS prototype', Technical report, SOCS Consortium, (2003). Deliverable D9.

[2] 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).

[3] M.E. Bratman, D.J. Israel, and M.E. Pollack, 'Plans and resource-bounded practical reasoning', *Computational Intelligence*, **4**, (1988).

[4] F. M. T. Brazier, B. Dunin-Keplicz, J. Treur, and R. Verbrugge, 'Modelling internal dynamic behaviour of BDI agents', in *ModelAge Workshop*, pp. 36–56, (1997).

[5] G. Brewka, 'Reasoning with priorities in default logic', in *Proceedings of AAAI-94, pp. 940-945*, (1994).

[6] N. Demetriou and A. C. Kakas, 'Argumentation with abduction', in *Proceedings of the fourth Panhellenic Symposium on Logic*, (2003).

[7] Y. Dimopoulos and A. C. Kakas, 'Logic programming without negation as failure', in *Logic Programming, Proceedings of the 1995 International Symposium, Portland, Oregon*, pp. 369–384, (1995).

[8] G. De Giacomo, H. J. Levesque, and S. Sardia, 'Incremental execution of guarded theories', *ACM Transactions on Computational Logic*, **2**(4), 495–525, (October 2001).

[9] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J. Ch. Meyer, 'Agent programming in 3APL', *Autonomous Agents and Multi-Agent Systems*, **2(4)**, 357–401, (1999).

[10] J. Jaffar and M. J. Maher, 'Constraint logic programming: A survey', *Journal of Logic Programming*, **19/20**, 503–581, (1994).

[11] A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni, 'A logic-based approach to model computees', Technical report, SOCS Consortium, (2003). Deliverable D4.

[12] A. C. Kakas and F. Sadri (Eds), *Computational Logic: Logic Programming and Beyond*, number 2407 and 2408 in LNAI, Springer-Verlag, 2002.

[13] A. C. Kakas, R. A. Kowalski, and F. Toni, 'The role of abduction in logic programming', in *Handbook of Logic in Artificial Intelligence and Logic Programming*, eds., D. M. Gabbay, C. J. Hogger, and J. A. Robinson, volume 5, pp. 235–324. Oxford University Press, (1998).

[14] A. C. Kakas, E. Lamma, P. Mancarella, P. Mello, K. Stathis, and F. Toni, 'Computational model for computees and societies of computees', Technical report, SOCS Consortium, (2003). Deliverable D8.

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

[16] J. A. Leite, J. J. Alferes, and L. M. Pereira, '$\mathcal{MINERVA}$: A dynamic logic programming agent architecture', in *Intelligent Agents VIII: 8th International Workshop, ATAL 2001, Seattle, USA*, volume 2333 of *LNAI*, pp. 141–157, (2002).

[17] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl, 'GOLOG: A logic programming language for dynamic domains', *Journal of Logic Programming*, **31**(1-3), 59–83, (1997).

[18] H. Prakken and G. Sartor, 'Argument-based extended logic programming with defeasible priorities', *J. of Applied Non-Classical Logics*, **7(1)**, (1997).

[19] A. S. Rao, 'AgentSpeak(L): BDI agents speak out in a logical computable language', in *Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96*, ed., Rudy van Hoe, volume 1038 of *LNCS*, pp. 42–55. Springer-Verlag, (1996).

[20] A. S. Rao and M. P. Georgeff, 'Modeling rational agents within a BDI-architecture', in *Readings in Agents*, eds., Michael N. Huhns and Munindar P. Singh, 317–328, Morgan Kaufmann Publishers, San Francisco, CA, USA, (1997).

[21] Y. Shoham, 'Agent-oriented programming', *Artificial Intelligence*, **60**(1), 51–92, (1993).

[22] K. Stathis, A. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali, 'PROSOCS: a platform for programming software agents in computational logic', in *Proc. of "From Agent Theory to Agent Implementation", AT2AI-4*, eds., J. Müller and P. Petta, Austria, (2004).

[23] S. R. Thomas, 'The PLACA agent programming language.', in *Intelligent Agents*, eds., M. J. Wooldridge and N. R. Jennings, Berlin, (1995). Springer-Verlag.

[24] M. Wooldridge, *An Introduction to Multiagent Systems*, John Wiley and Sons, 2002.