Ambient Intelligence Using KGP Agents

Kostas Stathis[1] and Francesca Toni[2]

[1] Department of Computing, City University London, UK
kostas@soi.city.ac.uk
[2] Department of Computing, Imperial College London, UK
ft@doc.ic.ac.uk

Abstract. We investigate the application of a logical model of agency, known as the KGP model, to develop agents for ambient intelligence applications. Using a concrete scenario, we illustrate how the logical formalism employed by a KGP agent allows a person to access the surrounding ambient through the agent in a transparent manner. We evaluate our claims by implementing the resulting interactions in PROSOCS, a prototype multi-agent systems platform that allows KGP agents to be deployed as components of ambient intelligence applications.

1 Introduction

The vision of Ambient Intelligence (AmI) is a society based on unobtrusive, often invisible interactions amongst people and computer-based services in a global computing environment [1]. Services in AmI will be ubiquitous in that there will be no specific bearer or provider but, instead, they will be associated with all kinds of objects and devices in the environment [7], which will not bear any resemblance to computers. People will interact with these services through intelligent and intuitive interfaces embedded in these objects and devices, which in turn will be sensitive to what people need.

For a large class of the envisaged AmI applications, the added value of these new services is likely to be for ordinary people, and more specifically on what people are trying to do in ordinary social contexts. Such a requirement begs for technologies that are *transparent*. Transparency is here interpreted broadly as the ability of a person to understand, if necessary, the functional behaviour of an object or device in the environment. Put simply, transparency should bring AmI interactions closer to the way people are used to think rather than the way machines operate. AmI applications, as a result, will need the ability to process a form of logic with computational features in order to make interactions between people and objects transparent not only at the interface level.

Another challenge posed by the AmI vision is that the electronic part of the ambient will often need to act intelligently on behalf of people. AmI cannot afford to stay within a direct manipulation interface where user requests result in monolithic system responses. Rather, the challenge here is that the ambient or, more specifically conceptual components of it, will often need to be not simply spontaneous but also proactive, behaving as if they were *agents* that act on behalf of people. It would be more natural, in other words, to use the *agent metaphor* in order to understand components of an intelligent ambient. An agent in this context can be a software (or hardware) entity that can sense or affect the environment, has knowledge of the environment and its own goals, and can pro-actively plan to achieve its goals or those of its user(s), so that the combined interactions of the electronic and physical environment provide a useful outcome to one or more people.

The contribution of this work is that it investigates the application of a logical model of agency, known as the KGP model, to build applications with the aim of providing a more transparent AmI. Agents in this context offer intelligent services to their users and manage spontaneous, ad-hoc interactions over distributed networks embedded in physical environments. The specific contribution of the work is that it shows how the KGP model of agency can be programmed in order to support the transparency and agency which might be required by AmI. The implementation of an application is discussed using PROSOCS, a multi-agent system platform that combines computational logic tools and Peer-to-Peer (P2P) Computing to support the envisaged AmI interactions.

2 KGP Agents

KGP (**K**nowledge, **G**oals and **P**lan) is a model of agency intended to represent the *internal* or *mental* state of an agent in a logical manner. The internal state of a *KGP* agent is a triple $\langle KB, Goals, Plan \rangle$, where:

- **KB** describes what the agent knows of itself and the environment and consists of separate modules supporting the different reasoning capabilities. For example, KB_{plan} supports planning, KB_{gd} goal decision, etc. One part of the KB, called KB_0 reflects changes in the environment and is typically updated by the agent when it observes the environment through its sensing capability. We will assume that KB_0 is the only part of the knowledge of the agent that changes over time.
- **Goals** is a set of properties that the agent has decided that it wants to achieve by a certain time possibly constrained via some temporal constraints. *Goals* are split into two types: *mental goals* that can be planned for by the agent and *sensing goals* that can not be planned for but only sensed to find out from the environment whether they hold or not.
- **Plan** is a set of "concrete" actions, partially time–ordered, of the agent by means of which it plans (intends) to satisfy its goals, and that the agent can execute in the right circumstances. Each action in *Plan* relies upon preconditions for its successful execution. These preconditions might be checked before the actions are executed.

A brief description of the transitions shown in Figure 1 is provided below:

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

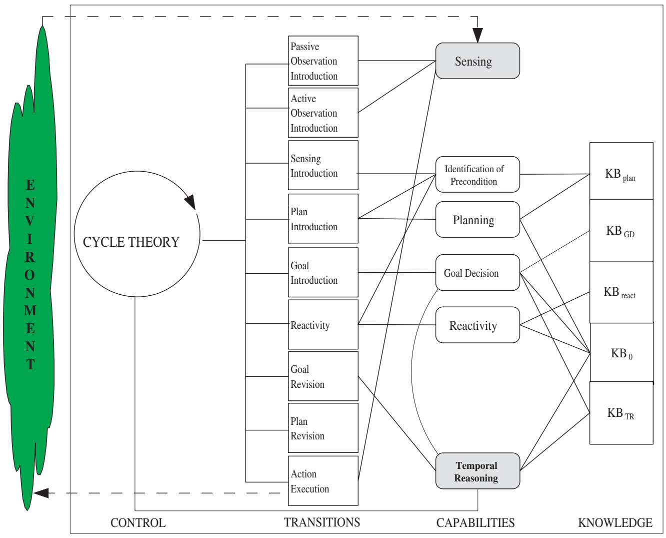


Fig. 1. In the *KGP* Model of Agency the *knowledge* in the agent's "mind" is operated by a set of *reasoning capabilities* that allow the agent to perform *planning, temporal reasoning, identification of preconditions of actions, reactivity* and *goal decision*, together with a *sensing capability* for the agent to perceive the environment in which it is situated. The capabilities are used by a set of *transition rules* describing how the mental state changes as a result of the agent being situated in an environment. Agent behaviour is obtained by combining sequences of transitions determined by reasoning with a *cycle theory* that controls this behaviour in a flexible manner.

- *Active Observation Introduction* (AOI) changes KB_0 by introducing the outcome of sensing actions for properties of interest to the agent; these properties are actively sought. 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 As computed by Planning, with the set of preconditions for the successful execution of A.
- *Goal Introduction* (GI) changes the *Goals* of a state by replacing them with goals that the Goal Decision capability decides to have highest priority.
- *Reactivity* (RE) is responsible for updating the current state of the agent by adding the goals and actions returned by the Reactivity capability. As with PI, this transition too uses the Identification of Preconditions capability, in order to equip each action A in the set As 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.

The model allows to deal with partial, dynamically evolving information, which is reasoned upon both via the Planning, Reactivity, and Temporal Reasoning capabilities. For more details on the KGP model the reader is referred to [4].

3 An AmI Scenario Based on KGP Agents

> Francisco Martinez travels from Spain to Italy. He makes his trip easier by carrying with him a personal communicator, a device that is a hybrid between a mobile phone and a PDA. The application running on this personal communicator provides the environment for an agent, a piece of software that augments the direct manipulation interface of the device with proactive information management within the device and flexible connectivity to smart services, assumed to be available in objects available in the global environment Francisco travels within. When Francisco arrives in Rome, the agent avoids a situation where Francisco has to unnecessarily buy a museum ticket, helps Francisco to find other people to share a taxi going from a hotel to a train station, and orders the taxi by holding a reverse auction to obtain the minimum price possible.

3.1 Reasoning Features of KGP Agents

To build the mental state of a KGP agent for a scenario like the one presented above we use well-understood reasoning techniques. We rely upon Abductive Logic Programming (and in particular the work described in [3]) to support the reasoning required for the capabilities of planning, reactivity, and identification of preconditions of actions. We also use Logic Programming with Priorities to support the generic functionality required for the goal decision capability and the development of cycle theories that control the agent's behaviour. Here, we focus on the specification of a KGP agent that reacts and plans in the environment in which it is situated. In Figure 2 we present a specialised version of the *event calculus* [5,8] that a KGP agent uses, appropriate to fit the KGP mode of operation (see [6] for more details).

In the example programs that follow we will give only the domain-dependent components of the knowledge base, namely (and typically) the definitions of the event calculus predicates *initially*, *initiates*, *terminates* and *precondition*, as

$$
\begin{aligned}
&\mathit{holds\_at}(F,T_2) \leftarrow && \mathit{happens}(O,T_1), \mathit{initiates}(O,T_1,F),\\
& && T_1 < T_2, \neg\ \mathit{clipped}(T_1,F,T_2).\\
&\mathit{holds\_at}(\neg F,T_2) \leftarrow && \mathit{happens}(O,T_1), \mathit{terminates}(O,T_1,F),\\
& && T_1 < T_2, \neg\ \mathit{declipped}(T_1,F,T_2).\\
&\mathit{holds\_at}(F,T) \leftarrow && \mathit{initially}(F), 0 < T, \neg\ \mathit{clipped}(0,F,T).\\
&\mathit{holds\_at}(\neg F,T) \leftarrow && \mathit{initially}(\neg F), 0 < T, \neg\ \mathit{declipped}(0,F,T).\\
&\mathit{clipped}(T_1,F,T_2) \leftarrow && \mathit{happens}(O,\ T), \mathit{terminates}(O,T,F), T_1 \leq T < T_2.\\
&\mathit{declipped}(T_1,F,T_2) \leftarrow && \mathit{happens}(O,\ T), \mathit{initiates}(O,T,F), T_1 \leq T < T_2.\\
&\mathit{clipped}(T_1,F,T_2) \leftarrow && \mathit{observed}(\neg F,T), T_1 \leq T < T_2.\\
&\mathit{declipped}(T_1,F,T_2) \leftarrow && \mathit{observed}(F,T), T_1 \leq T < T_2.\\
&\mathit{holds\_at}(F,T_2) \leftarrow && \mathit{observed}(F,T_1), T_1 \leq T_2, \neg\ \mathit{clipped}(T_1,F,T_2).\\
&\mathit{holds\_at}(\neg F,T_2) \leftarrow && \mathit{observed}(\neg F,T_1), T_1 \leq T_2, \neg\ \mathit{declipped}(T_1,F,T_2).\\
&\mathit{happens}(O,T) \leftarrow && \mathit{executed}(O,T).\\
&\mathit{happens}(O,T) \leftarrow && \mathit{observed}(O,T',T).\\
&\mathit{happens}(O,T) \leftarrow && \mathit{assume\_happens}(O,T).\\
&\mathit{holds\_at}(F,T) \leftarrow && \mathit{assume\_holds\_at}(F,T).
\end{aligned}
$$

Integrity Constraints: $\mathit{holds\_at}(F,T), \mathit{holds\_at}(\neg F,T) \Rightarrow \mathit{false}.$
$\mathit{assume\_happens}(A,T), \mathit{precondition}(A,P) \Rightarrow \mathit{holds\_at}(P,T).$

Fig. 2. Extract of how a KGP agent situated in an environment concludes that a fluent F holds at a time T. Fluents holding at the initial state are represented by what holds *initially*. Fluents are *clipped* and *declipped* by events happening at different times. Events that happen will *initiate* and *terminate* properties holding at or between times. Fluents will also hold if they have been *observed* or result from actions that have been *executed* by the agent. An operation can be made to happen and a fluent can be made to hold simply by assuming them. In KGP we also use domain-independent integrity-constraints, to state that a fluent and its negation cannot hold at the same time and, when assuming (planning) that some action will happen, we need to enforce that each of its preconditions hold.

well as any other definitions of interest. In this context we will assume that (timed) action operators are represented within the (*Plan* component of the) state of an agent in the form:

$\mathit{op}(arg_1, arg_2, ..., arg_n, T)$

where *op* is the operator of the action, T the time argument of the action, and $arg_1, ..., arg_n$ any other (ground) arguments for the action. Also, we will assume that actions are represented within KB_{plan}, KB_{react}, KB_{TR} as:

$\mathit{assume\_happens}(\mathit{op}(arg_1, arg_2, ..., arg_n), T).$

Similarly for timed fluent literals. Furthermore, we will give KB_0 as a set of facts of the form:

$\mathit{executed}(\mathit{op}(arg_1, arg_2, ..., arg_n, T), \tau).$
$\mathit{observed}(\mathit{lit}(arg_1, arg_2, ..., arg_n, T), \tau).$
$\mathit{observed}(\mathit{op}(arg_1, arg_2, ..., arg_n, \tau 0), \tau).$

where *op* is an action operator, *lit* is a fluent literal, T is an existentially quantified variable, τ, $\tau 0$ are ground constants (the concrete time of execution or observation). However, we will assume that the capabilities of planning, reactivity and temporal reasoning (and thus goal decision that uses temporal reasoning), use the ground variant of KB_0, namely the set of facts of the form

$executed(op(arg_1, arg_2, ..., arg_n, \tau))$.
$observed(lit(arg_1, arg_2, ..., arg_n, \tau))$.
$observed(op(arg_1, arg_2, ..., arg_n, \tau 0), \tau)$.

Finally, we will assume that the behaviour of agents is given via operational traces, where passive observations from the environment (via POI) is always treated as some sort of interrupt.

3.2 Not Buying a Museum Ticket

Francisco has asked his agent to buy a museum ticket for some (state-run) museum that Francisco wants to visit. The agent has developed a plan to buy it, however, before executing the plan the agent observes that it is European Heritage Day (ehd) from a “message” sent by another agent *ma* (the museum agent) and, as a result, all state-run museums in Europe are free that day. Based on this observation its goal is achieved, the buying action is not executed, and both goal and plan are deleted.

State. The agent has a goal *g1* to buy a ticket within a specific time–limit, as well as a plan to buy it by executing an action *a1*:

$Goals = \{g_1 = \langle have(ticket, T), \bot, \{T \leq 10\}\rangle,$
$Plan = \{a_1 = \langle buy(ticket, T'), g_1, have\_money(T'), \{T' < T\}\rangle.$

KB_{TR} contains:

$initiates(ehd, T, have(ticket))$.
$initiates(buy(O), T, have(O))$.
$precondition(buy(O), have\_money)$.

The remaining components of KB do not play any role in this example and can be considered to be empty.

Trace. Francisco can eventually trace the behaviour of the agent (the short names for transitions POI, GR and PR below have already been described in section 2):

POI: $observed(ma, ehd(Time), 6)$.
GR: at time 8, g_1 is eliminated from *Goals* as it achieved.
PR: at time 10, a_1 is eliminated from *Plan*, as g_1 is not in *Goals* anymore.

3.3 Sharing a Taxi

Francisco's agent tries to find a person that is likely to accept to share a taxi with Francisco from the hotel to the train station. Francisco's agent has asked all other available agents in the hotel society for the resource, but did not manage to get it, so it suspends its goal. But, when a new agent becomes available in the hotel, then Francisco's agent can go back and ask the new agent.

State. The agent has a goal to reach the station and no plan for it:

$Goals = \{g_1 = \langle reach\_station(T), \bot, \{T < 20\}\rangle$, $Plan = \{\}$.

KB_{plan} consists of the following domain-dependent knowledge:

$holds\_at(reach\_station, T) \leftarrow$
$\quad holds\_at(accepted\_share, T'), T' < T.$
$holds\_at(accepted\_share, T) \leftarrow$
$\quad holds\_at(customer(O), T'),$
$\quad happens(tell(f, O, request(share\_taxi), d(ex7, T')), T'),$
$\quad holds\_at(accepted\_request(O, share\_taxi), T''),$
$\quad T' < T'' < T.$

$precondition(tell(f, O, request(share\_taxi), D), \neg already\_rejected(O))$
$initiates(tell(O, f, reject(request(share\_taxi)), D),\ T,$
$\quad already\_rejected(O))$
$initiates(tell(O, f, accept(request(share\_taxi)), D),\ T,$
$\quad accepted\_request(O, share\_taxi)).$

These definitions basically impose to ask one after the other, in sequence, agents of customers of the hotel, without asking the same agent twice, until one accepts the offer. Knowledge about customers in the hotel can be drawn from the following domain-dependent definitions, which is a simplification of the ones found in [10], where to access the electronic environment of a hotel the agent joins an *artificial agent society* residing in the hotel's AmI infrastructure:

$initially(customer(o1)).$
$initially(customer(o2)).$
$initially(\neg already\_rejected(o1)).$
$initially(\neg already\_rejected(o2)).$
$initiates(join(C), T, customer(C)).$

To join an artificial society the agent executes the action of joining itself:

$executable(join(f), T)).$
$executable(tell(f, C, S, D)) \leftarrow C \neq f.$
$assume\_happens(A, T), not\, executable(A) \Rightarrow false.$

As before, we assume that KB_0 is initially empty and that the other components of KB do not play any role (they can be considered empty too).

Trace. The internal transitions of the agent can be summarised as follows:

PI: action a_{o1} and goal g_{o1} are added to $Plan$ and $Goals$, respectively:
$a_{o1} = \langle tell(f, o1, request(share\_taxi), d(ex7, T'), T'), g_1, \neg already\_rejected(o1), \{T' < T''\}\rangle.$
$g_{o1} = \langle accepted\_request(o1, share\_taxi, T''), g_1, \{T'' < T\}\rangle.$

AE: a_{o1} is executed - $executed(tell(f, o_1, request(share\_taxi), d(ex7, T'), T'), 1)$.
POI: $observed(o_1, tell(o_1, f, reject(request(share\_taxi)), d(ex7, T'), 3), 3)$.
GR & PR: g_{o1} and a_{o1} are deleted from *Goals* and *Plan*, respectively.
PI: action a_{o2} and goal g_{o2} are added to *Plan* and *Goals*, respectively:
$a_{o2} = \langle tell(f, o2, request(share\_taxi), d(ex7, T^*), T^*), g_1, \neg already\_rejected(o1), \{T^* < T^{**}\}\rangle$.
$g_{o2} = \langle accepted\_request(o2, share\_taxi), T^{**}), g_1, \{T^{**} < T\}\rangle$.
AE: a_{o2} is executed - $executed(tell(f, o_2, request(share\_taxi), d(ex7, T^*), T^*), 5)$.
POI: $observed(o_2, tell(o_2, f, reject(request(share\_taxi)), d(ex7, T^*), 6), 6)$.
GR & PR: g_{o2} and a_{o2} are deleted from *Goals* and *Plan*, respectively.
POI: new agent joins $observed(o3, join(o3, 3), 10)$.
PI: action a_{o3} and goal g_{o3} are added to *Plan* and *Goals*, respectively:
$a_{o3} = \langle tell(f, o3, request(share\_taxi), d(ex7, T^!), T^!), g_1, \neg already\_rejected(o3), \{T^! < T^{!!}\}\rangle$.
$g_{o3} = \langle accepted\_request(o3, share\_taxi), T^{!!}), g_1, \{T^{!!} < T\}\rangle$.
AE: a_{o3} is executed - $executed(tell(f, o3, request(share\_taxi), d(ex7, T^!), T^!), 12)$.
POI: $observed(o_3, tell(o_3, f, accept(request(share\_taxi)), d(ex7, T^!), 15), 15)$.
GR and PR: g_{o3}, g_1 and a_{o3} are deleted from *Goals* and *Plan*, respectively.
The presence of the new agent has allowed f to achieve its goal successfully.

3.4 Ordering a Taxi

In order to get a taxi with the lowest fare, Francisco's agent f decides to hold a reverse auction, where the agent sets the maximum price Francisco is willing to pay, starts the auction by broadcasting to all available taxis in the ambient, and after collecting all bids from the different taxis, it notifies the winner (namely the agent representing the taxi offering the lowest fare) as well as all the other bidders (namely the other agents representing the rest of the taxis who offered higher fares than that accepted).

Bidder agents. The way taxi agents that act as bidders react is described in the $KB^{t_1}_{react}$, $KB^{t_2}_{react}$ and $KB^{t_3}_{react}$, all containing the rule:

$$\begin{array}{l} observed(tell(A, t_i, openauction(taxi, TEnd, TDeadline), Id, TOpen), \\ holds\_at(cost\_taxi(C), TOpen) \\ \Rightarrow \\ assume\_happens(tell(t_i, A, bid(I, C), Id), T), \\ TOpen < T < TEnd. \end{array}$$

In other words, if somebody opens an auction asking for a taxi, then reply with a bid. We assume that the destination of the journey is not an issue in this example, and that each of the taxi agent will have a concrete cost in their KB, e.g. $KB^{t_1}_{react}, KB^{t_2}_{react}, KB^{t_3}_{react}$ would have, respectively:

$initially(cost\_taxi(3))$.
$initially(cost\_taxi(5))$.
$initially(cost\_taxi(7))$.

Auctioneer agent. In formulating the knowledge base of f, we will assume the existence of an *external object* that we will refer to as ac (standing for *auction calculator*) which f will interact with in order to get a solution to the (simple) optimisation problem it has to solve in order to decide the winner of the auction that f will open. We will not make any assumptions here on the internal structure of ac, but we will assume that it is "wrapped" in such a way that f can communicate with it. Moreover, we will assume that f and ac are synchronized, in the sense of sharing the same clock, and that ac is committed to return a solution of the optimisation problem by the auction deadline that f passes to it. The following is the set of reactive rules in KB^f_{react}:

$holds\_at(want\_taxi, T),$
$holds\_at(available\_taxi(X), T)$
$\Rightarrow$
$assume\_happens(tell(f, X, openauction(taxi, T+5, T+10), d(fpsb, T)), T).$

In other words, whenever in need of a taxi, f opens an auction for a taxi, communicating to all taxi drivers it believes to be available the end time of the auction ($T+5$) and the deadline or deciding the winner of the auction ($T+10$).

$observed(C, tell(C, f, bid(I, C), Id, T0), T),$
$executed(f, tell(f, C, openauction(I, TEnd, TDeadline), Id, T), T1),$
$T1 < T0$
$\Rightarrow$
$assume\_holds(tell(f, ac, calculate(Id, I, TEnd, TDeadline), d(ac, T')), T'),$
$T < T' < TDeadline - 2.$

Based on this rule, agent f sends every bid it receives to the auction calculator ac, as soon as it receives it.

$observed(ac, tell(ac, f, results(Id, I, TEnd, TDeadline, (C, Q), R), T0), T)$
$executed(f, tell(f, X, openauction(I, TEnd, TDeadline), Id, T), T1),$
$T1 < T0$
$\Rightarrow$
$assume\_holds(tell(f, C, answer(win, I, Q), Id), T'), T' < TDeadline.$

With the rule above, agent f is in a position to notify the winner of the auction as soon as it receives the decision from ac (we assume here that ac sends the notification by the deadline of the auction and thus f is able to meet this deadline in turn).

$observed(ac, tell(ac, f, results(Id, I, TEnd, TDeadline, (C, Q), R), T0), T)$
$executed(f, tell(f, X, openauction(I, TEnd, TDeadline), Id), T),$
$T1 < T0$
$\Rightarrow$
$assume\_happens(tell(f, C, answer(lose, I, Q), Id), T),\ TEnd < T < TDeadline.$

Similarly, agent f is now in a position to notify the losers of the auction as soon as it receives the decision from ac using its KB^f_{react}:

$initially(available(t_1))$.
$initially(available(t_2))$.
$initially(available(t_3))$.

We expect that the auction calculator will use the steps:

step 1: $findall\_bids(Id, I, TEnd, ValidBids)$,
step 2: $find\_minimum(ValidBids, (Winner, Bid))$,
step 3: $Lost = minus(ValidBids, (Winner, Bid))$.

so that, all the bids are computed ($ValidBids$), then the winning bid is determined (the tuple ($Winner, Bid$)), and then the list of bids that remained ($Lost$) is evaluated.

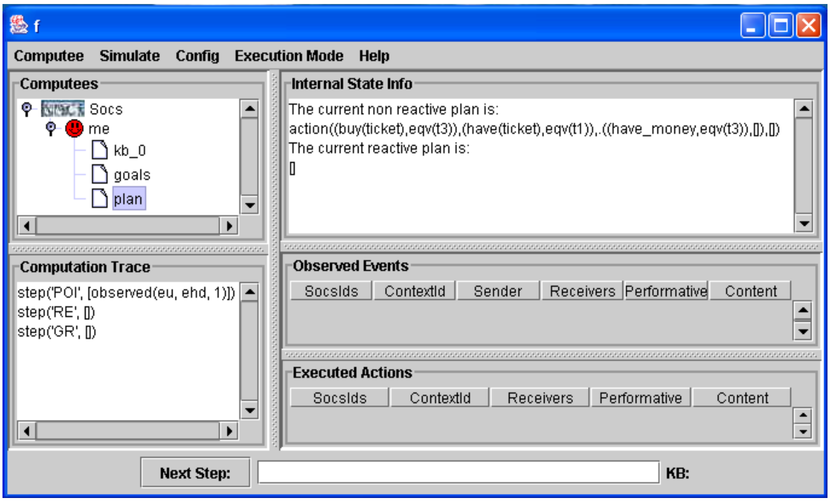
Auction Interactions. To save space, instead of giving the detailed traces of each agent participating in the auction, we present here a summary of the interaction between the auctioneer and the taxi agents. The auctioneer (f) generates (via RE) and executes (via AE) an action to open the auction. Then the bidders (taxis) record the communicative act via POI and put forward their bids (offering fares of 3, 5, 7, respectively), again by RE and AE. Once all taxi agents send their bids, via POI, f records the bids and, via RE, it sends them to the *ac*. After having received the final calculations from *ac*, recorded via POI, f generates, via RE, the communicative actions to make t_1 a winner and the other two losers, and, via AE, it executes the actions.

3.5 Implementation Issues

We have implemented the interactions that we have discussed in the previous section using PROSOCS [9], an experimental platform that allows the deployment KGP agents in open computing environments such those characterising AmI. We have been using the platform's basic interface presented in Fig. 3 to show whether a KGP agent will produce the expected behaviour. Using this basic interface we can inspect the various parts of the behaviour of the agent and how the computational trace of the agent can be examined. An addition advantage of the PROSOCS platform is that it allows new deployed agents in a network to be discovered automatically, using the P2P protocols available in the JXTA project [11]. JXTA makes PROSOCS a suitable platform for AmI applications, especially those that require ad–hoc networks for their development. PROSOCS also provides a society infrastructure [2], that allows interactions between agents to be checked upon whether they conform to specific *social rules.*

4 Concluding Remarks

We have illustrated how to apply the KGP model of agency to develop ambient intelligence applications. We have shown how the logical formalism employed by


f
Computee Simulate Config Execution Mode Help
Computees
Socs
me
kb_0
goals
plan
Internal State Info
Computation Trace
Observed Events
SocsIds ContextId Sender Receivers Performative Content
Executed Actions
SocsIds ContextId Receivers Performative Content
Next Step:
KB:

```
The current non reactive plan is:
action((buy(ticket),eqv(t3)),(have(ticket),eqv(t1)),.((have_money,eqv(t3)),[]),[])
The current reactive plan is:
[]
```

```
step('POI', [observed(eu, ehd, 1)])
step('RE', [])
step('GR', [])
```

Fig. 3. A screen-shot of the agent interface supported in PROSOCS. A KGP agent in PROSOCS is called a *computee*, to underline the fact that the the mind of the agent is developed in computational logic. On the top-left part of the figure the components of the mental state of the agent are shown, e.g. the KB_0, the goals and the plan. Depending on which one of these components is selected, the content of the component is shown in the top-right part of the figure. The computational trace is shown in the bottom-left part, while any observed events or executed actions are shown in the middle- and bottom-left part of the screen.

a KGP agent allows a person to access the surrounding ambient in a transparent manner. Transparency in KGP is the agent's ability to provide a narrative of its mental state, in terms of transitions, that allow a person to understand the agent's behaviour in achieving a goal. We have shown how three concrete examples can be formulated in the model and we have briefly discussed their implementation in the prototype platform PROSOCS.

Some aspects of the KGP model both generally and for Ambient Intelligence applications will require further investigation. For example, we have not addressed to date the issue of endowing KGP agents with learning capabilities which would allow to revise and augment suitability their knowledge base. We envisage that an additional *Knowledge Revision* transition could be smoothly incorporated into the model, but its definition will require further thought. Also, we have not studied the problem of distinguishing between relevant and irrelevant knowledge when reasoning in order to perform a specific user task. We envisage that this issue will be of great practical importance towards the actual use of KGP agents for applications.

We are currently developing more detailed experiments with the aim to extend the platform to allow KGP agents to interact not only with agents but also with objects in the environment in a generic way. Additional experimentation, especially in the context of the reverse auction example, is under way. Also, the presentation of agent traces using a more "natural" interface is the subject of future work. To date, the experimentation has been based on PCs in a lab. Further experimentation in less controlled settings, e.g. wireless networks using PDAs, will be necessary to fully demonstrate the suitability of the model for Ambient Intelligence.

Acknowledgements. This work was funded by the IST programme of the EC, FET under the IST-2001-32530 SOCS project, within the GC proactive initiative, and by the Italian MIUR programme "Rientro dei Cervelli".

References

1. In P. Spirakis, A. Kameas, and S. Nikoletseas, editors, *Proc. International Workshop on Ambient Intelligence*, Santorini, Greece, June 2003. CTI Press.
2. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Compliance verification of agent interaction: a logic-based tool. In J. Müller and P. Petta, editors, *Proc. of the 4th International Symposium "From Agent Theory to Agent Implementation"*, Vienna, Austria, April 13-16 2004.
3. U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The CIFF proof procedure for abductive logic programming with constraints. In *Proc. of the 9th European Conference on Logics in Artificial Intelligence(Jelia04)*, 2004. To appear.
4. A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *Proc. of the European Conference on Artificial Intelligence (ECAI04)*, 2004. To appear.
5. R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.
6. P. Mancarella, F. Sadri, G. Terreni, and F. Toni. Partial planning for situated agents. In *Proc. of 5th Workshop on Computational Logic in Multi-agent Systems (CLIMA V)*, 2004. To appear.
7. B. Raducanu and P. Markopoulos. Applications of vision-based attention-guided perceptive devices to aware environments. In *Proc. of the 1st European Symposium on Ambient Intelligence*, pages 410–418. Springer-Verlag, 2003.
8. M. Shanahan. Event calculus planning revisited. In *Proc. 4th European Conference on Planning*, volume 1348 of *LNAI*, pages 390–402. Springer Verlag, 1997.
9. 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, *Proc. of the Fourth International Symposium "From Agent Theory to Agent Implementation"*, Vienna, Austria, April 13-16 2004.
10. F. Toni and K. Stathis. Access-as-you-need: a computational logic framework for flexible resource access in artificial societies. In *Proc. of the Third International Workshop on Engineering Societies in the Agents World (ESAW'02)*, LNAI. Springer Verlag, 2002.
11. B. Traversat, M. Abdelaziz, D. Doolin, M. Duigou, J. C. Hugly, and E. Pouyoul. Project JXTA-C:Enabling a Web of Things. In *Proc. of the 36th Hawaii International Conference on System Sciences (HICSS'03).* IEEE Press, 2003.