Agent Planning, Negotiation and Control of Operation

Antonis C. Kakas[1] **Paolo Torroni**[2] and **Neophytos Demetriou**[1]

Abstract. This paper presents a framework that integrates three aspects of agency: *planning*, for proactive behaviour, *negotiation*, for social behaviour and resource achievement, and *control of operation*, for reconciling rationality with reactivity. Agents are designed and programmed in a computational logic-based language where these aspects are accommodated in a declarative and modular way. We show how this framework can be applied to agent problems requiring negotiation and resource achievement and present some of its formal properties. The framework can be implemented based on a communication platform for agent interaction and on well-established logic programming technologies for agent reasoning.

1 Introduction

Agents often need to collaborate (operate jointly) with other agents in order to achieve their goals. This need for collaboration presents a challenging problem of synthesizing together several different agent tasks, such as *planning* - including the task of recognizing if and when a collaboration is needed, *negotiation* with other agents for their needs and *execution* of plans. Within this synthesis agents also need to take important decisions as to which plan to select in order to achieve their goals, which agents to choose to negotiate with and how to negotiate given the agent's own characteristics and any protocols that they are meant to comply with.

Many logic-based agent frameworks have been proposed to express agent reasoning [10], dialogue and negotiation [1, 15], planning and integration with data sources [2], focussing on a particular aspect of the picture at a time . We present a fully-fledged framework for designing agents that can operate jointly in problem solving. This framework is based on a *BDI*-like model of agency [14]: the KGP model developed in the SOCS project [9], where an agent is separated into modular components expressed in the high-level declarative language of Computational Logic. We extend it by allowing preference policies in any knowledge component of the agent and conditional plans in its planning capability, and by supporting negotiation between agents with a variety of individual behaviours, e.g. cooperative or not. We study how planning, negotiation and temporal reasoning about changes in the world can be synthesized together to set up and carry out collaborative operations of agents. This modular synthesis facilitates the application to a wide class of problems, and it allows adaptability of behaviour to changing conditions, e.g. relative roles of agents, in an open and dynamically changing environment.

A key element for this is the fact that in the design of an agent its overall control of operation and its various components contain *private preference policies* that shape the various decisions of the agent at different levels, e.g decisions of which plan to use, which agents to collaborate with, which way to reply to requests. We illustrate this flexibility of the approach by giving an example of an agent's private policy for the decision of how to carry out a negotiation conversation and show how agents can hold a negotiation dialogue, using such policies, guaranteeing various formal properties of this operation. A prototypical implementation of the framework exists [17], with application to example problems proposed in the literature [13, 4].

[1] Department of Computer Science, University of Cyprus, Nicosia, Cyprus email: antonis@cs.ucy.ac.cy, k2pts@cytanet.com.cy
[2] DEIS, Università di Bologna, Bologna, Italy email: paolo.torroni@unibo.it

2 Agent Problem Solving with Collaboration

We will adopt a *BDI*-like model of agency [14], where an agent has $Knowledge$, $Goals$ and $Plans$ for these goals. Its knowledge, *KB*, consists of several *modular* components of separate concerns. Such components are KB_{Plan}, KB_{TR} and KB_{Neg} for planning, temporal reasoning and negotiation respectively. The language of representation and computation is that of *Computational Logic* (CL) [12, 11] and in particular, Abductive Logic Programming (ALP) [7] together with Logic Programming with Priorities (LPP) [10, 3].

The agent has a state, $S = \langle KB_0, Goals, Plan \rangle$, where:

- KB_0 is a logic program holding facts about actions that have occurred and observations of fluent properties in the world;
- $Goals$ consists of goals of the form $holds(L, T)$ expressing the desire that "a fluent literal L is true at time T";
- $Plan$ is a (conditional) plan for $Goals$ given KB_0 expressing the current intentions of the agent about how to satisfy its goals.

Abductive reasoning in ALP is used as the basis for planning and other tasks that may require reasoning with incomplete information. Reasoning with priorities in LPP gives a form of preference reasoning that affects the decision making of an agent. Any component of the knowledge base can contain a *preference policy* that gives in a declarative way a selection mechanism sensitive to the current conditions of operation, e.g., in KB_{Plan} which plan to choose, or in KB_{Neg} which agent to choose to negotiate with.

The operation of an agent is regulated via a declarative *cycle theory*, also expressing in LPP as a preference policy of internal state transitions. This policy determines the next state transition as a preferred transition according to some general behaviour characteristics expressed by the cycle theory. Cycle theories thus allow for the flexibility to capture at a high level different profiles of behaviour for the agents, e.g. cooperative or non-cooperative etc.

2.1 Planning and Temporal Reasoning

Given a (set of) goal(s), an agent is able to generate a number of alternative plans to achieve them, using the Abductive Event Calculus (AEC) [6, 16] as its basis for planning. In fact, the agent needs to perform *conditional planning* in view of the open environment in which it operates. In the AEC, a conditional plan P for a goal G is a triple $P = \langle \Delta, \mathcal{C}, \mathcal{O} \rangle$, where Δ is a set of timed actions, $\mathcal{C}$ a set of timed conditions on a particular set of fluents, and $\mathcal{O}$ a partial order on the times of actions and conditions. This is required to satisfy

$$T_{AEC} \cup \mathcal{C} \cup \Delta \models G \ and \ T_{AEC} \cup \mathcal{C} \cup \Delta \models I_{AEC}$$

where T_{AEC} contains the general theory of AEC and the current *KB*$_0$, and I_{AEC} are its integrity constraints. T_{AEC} also contains the agent-depended knowledge, mainly as a set of *init(iates)* and *term(inates)* clauses, expressing the agents particular capabilities to perform actions in order to initiate or terminate properties. For example, an agent could use the clause: $init(play(CD), T, music)$, to plan the action "$play(CD)$" and reach the goal "(listen to) $music$".

The conditions in $\mathcal{C}$ are on particular fluents representing properties that agents cannot bring about by their own actions alone. An example is the fluent $promise(\textit{from}(\textit{Peer}), \textit{Res})$, representing the property that agent *Peer* has promised to give away the resources *Res*. These conditions link planning and negotiation together, as follows. To perform an action an agent will typically need to *use* resources. In planning, this is modelled as a fluent condition *use_of(Res)* to mean that resources *Res* have been *allocated*. In the simple example above, the feasibility of the effect of "$play(CD)$" relies on the possibility to allocate two resources: a working CD player and a CD. This then is expressed by the following rule:

$init(playMusic, T, music) \leftarrow$
$holds(use\_of(workingCDP), T), holds(use\_of(CD), T).$

The general theory of allocating resources is captured by the following (domain independent) rules as part of the AEC theory, T_{AEC}.

$init(get(\textit{from}(\textit{Peer}), \textit{Res}), T, available(\textit{Res})).$
$precond(get(\textit{from}(\textit{Peer}), \textit{Res}), promise(\textit{from}(\textit{Peer}), \textit{Res})).$
$precond(get(\textit{from}(\textit{Peer}), \textit{Res}), asked(\textit{Peer}, \textit{Res})).$
$init(tell(\textit{Peer}, msg(\textit{from}(\textit{self}), req(\textit{Res}))), T, asked(\textit{Peer}, \textit{Res})).$
$init(use(\textit{Res}), T, use\_of(\textit{Res})).$
$term(use(\textit{Res}), T, available(\textit{Res})).$
$precond(use(\textit{Res}), available(\textit{Res})).$

Hence, if an agent needs the use of a resource *Res*, it can arrange in its plan to make *Res* available by *get*ting it from some other agent. The preconditions of this require that it must ask the agent and that it must have a promise for *Res*. For the first it will add in its plan a $tell(\textit{Peer}, msg(\textit{from}(\textit{self}), req(\textit{Res})))$ communication action to request this. The promise precondition is added in the conditions $\mathcal{C}$ of the plan. The request action when executed will start (as we will see below) a negotiation with the *Peer* that may or may not lead to the required promise. If it does then the agent will be able to execute the get action and acquire its need, otherwise the plan is not feasible. We do not model here the actual delivery of resources, and we assume that agents abide by their promises.

Promises are generated by successful negotiations dialogues. One way to capture this in the T_{AEC} theory is the following:

$init(tell(\textit{self}, msg(\textit{from}(\textit{Peer}), terms(\textit{Needs}, \textit{Terms}))), T,$
$cond\_promise(to(\textit{self}), terms(\textit{Needs}, \textit{Terms}))).$

where *cond_promise*$(to(\textit{Peer}), terms(\textit{Needs}, \textit{Terms}))$ means that the agent promises *Needs*, provided that the other agent promises *Terms* in exchange. A conditional promise and an unconditional promise of the *Terms* link together, to give an unconditional promise of the request. This is captured by the rules:

$init(tell(\textit{Peer}, msg(\textit{from}(\textit{self}), yes(\textit{Needs}, \textit{Terms}))), T,$
$promise(to(\textit{Peer}), \textit{Terms})) \leftarrow$
$holds\_at(cond\_promise(to(\textit{self}), terms(\textit{Needs}, \textit{Terms}), T).$
$init(tell(\textit{Peer}, msg(\textit{from}(\textit{self}), yes(\textit{Needs}, \textit{Terms}))), T,$
$promise(\textit{from}(\textit{Peer}), \textit{Needs})) \leftarrow$
$holds\_at(cond\_promise(to(\textit{self}), terms(\textit{Needs}, \textit{Terms}), T).$

We will assume that all needs in a plan are collected together and compared with the agent's initially available needs to compute its additional needs. These can then be requested together at the start of the plan, in an initial step, P^R, working under the simplified assumptions that agents negotiate for all their needs together, before starting to execute the plan, in order to ensure its feasibility.

2.2 Agent Negotiation Policies

An agent has several *private* preference policies modularly separated in its knowledge base components. We present here, as an example, an agent's *Negotiation Conversation* policy that it uses to choose its responses during a negotiation conversation. We will see below how this is integrated with a global negotiation protocol and how these are captured within the operation preference policy of the cycle theory of the agent.

The Negotiation Conversation policy is expressed by a theory, *KB*$_{NC}$, of rules and priorities on rules within the extended logic programming framework of $LPwNF$ [8, 10]. This framework is equipped with an argumentation-based notion of preference entailment, that we refer to as $\models_{pr}$. Intuitively, given a theory $\mathcal{T}$, $\mathcal{T} \models_{pr} L$ means that the literal, L, is a conclusion of a sub-theory of $\mathcal{T}$ which is "preferred", w.r.t. 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. Lack of space does not allows us to give more details and we will concentrate only to illustrate the flexibility afforded by the framework. A simple policy is as follows:

> "You can reject requests. You can accept a request if you can currently satisfy it. Prefer to get terms (your current needs) in exchange of accepting a request. Similarly, terms offered in exchange of satisfying a request can be rejected or accepted if they can be currently satisfied."

We will consider a 2-agent setting, where messages are received by an agent Ag in the form $msg(\textit{from}(Ag'), Content)$ and then added to Ag's *KB*$_0$. Then this policy can be represented with rules of the following form, where we have simplified several issues, such as response time, which are out of the main scope of this paper:

$r_{terms}(\textit{Peer}, terms(\textit{Needs}, \textit{Terms})) :$
$msg(to(\textit{Peer}), terms(\textit{Needs}, \textit{Terms})) \leftarrow$
$msg(from(\textit{Peer}), req(\textit{Needs})), current\_satisfiable(\textit{Needs}),$
$current\_terms(\textit{Terms}).$

This rule (here $r_{terms}(\textit{Peer}, terms(\textit{Needs}, \textit{Terms}))$ is a parameterized term that names the rule) provides an argument for the agent to accept a request for *Needs*, when these can be currently satisfied. This acceptance is under some *Terms* which are drawn from the needs of the agent in its current plan. If *Terms* is empty, r_{terms} means unconditional acceptance. Similarly, we have a rule, named $r_{reject}(\textit{Peer}, Request)$, for arguments to reject a request.

Beside these rules providing arguments to either reject or accept a request, the policy also contains priority rules that shape a preference policy for the agent. For example, the preference to accept with *Terms* over rejecting is captured by:

$R_{terms|reject}(\textit{Peer}, \textit{Needs}, \textit{Terms}) :$
$r_{terms}(\textit{Peer}, terms(\textit{Needs}, \textit{Terms})) > r_{reject}(\textit{Peer}, req(\textit{Needs})).$

where $Rule1 > Rule2$ is a special binary predicate in the representation language that expresses a priority over any two rules of the given theory named by the terms $Rule1$, $Rule2$.

This rule applies even in the case where the *Terms* are empty. But accepting a request with empty terms has no personal gain and so an agent may prefer to reject a request in such a case:

$R_{reject|terms}(\textit{Peer}, \textit{Needs}, \emptyset) :$
$r_{reject}(\textit{Peer}, req(\textit{Needs})) > r_{terms}(\textit{Peer}, terms(\textit{Needs}, \emptyset)).$

We can then distinguish two types of agents: *cooperative* and *non-cooperative*, by assigning different priority among these two priority rules. Hence including in the theory the *higher order* priority rule

C^{coop}(*Peer*, *Needs*, $\emptyset$) :
$R_{terms|reject}$(*Peer*, *Needs*, $\emptyset$) $>$ $R_{reject|terms}$(*Peer*, *Needs*, $\emptyset$).
gives a *cooperative* policy whereas for a *non-cooperative* policy we will include a rule $C^{non-coop}$ stating opposite priorities.

Given such a policy theory, KB_{NC}, and a current message, $msg(\textit{from}(\textit{Peer}), Content)$, the construction of a response message, $msg(to(\textit{Peer}), Reply)$, is given by:
$KB_{NC} \cup \{msg(\textit{from}(\textit{Peer}), Req)\} \models_{pr} msg(to(\textit{Peer}), Reply)$
where $\models_{pr}$ is the argumentation-based preference entailment that derives from the theory a conclusion with the strongest argument. For example, in the above policies given a current message of request for some *Needs* that are satisfiable, we have arguments both for accepting with terms and for rejecting. But, when *Terms* are not empty, the strongest is that of accepting, due to the priority rule $R_{terms|reject}$. Instead, if the current terms are empty, then the other priority rule $R_{reject|terms}$ also applies and so the argument of rejecting is equally strong. The strongest argument is then decided by which one of the higher-order rules (C^{coop} or $C^{non-coop}$) is in the theory.

The modular distinction of the two policies of cooperation and non-cooperation simply in terms of the addition of an extra rule, C^{coop}, or $C^{non-coop}$, respectively, shows the *flexibility* of the framework. This can be further illustrated by considering how a policy can be sensitive to the context of any particular negotiation, allowing the same agent to be cooperative in some cases and non-cooperative in other cases. For example, if we want an agent to be cooperative only when a request comes from a colleague we can we exploit the fact that priority rules can be *conditional* to capture this via the rule:
C^{coop}(*Peer*, *Needs*, $\emptyset$) :
$R_{terms|reject}$(*Peer*, *Needs*, $\emptyset$) $> R_{reject|terms}$(*Peer*, *Needs*, $\emptyset$) $\leftarrow$
$colleague$(*Peer*).

We could have more elaborated policies, accommodating for example dynamic events such as authorizations. We would then replace in the above rule the condition $colleague$ with $authority$ which becomes true dynamically when the society instructs so.

2.3 Control of Operation

The operation of our agents is regulated by their cycle theories expressing preference policies which encode declaratively different behaviour characteristics in the operation of the agents. For agents in this paper it will suffice to consider only a specific cycle theory, which can be compiled into a fixed cycle of operation. Informally, this cycle goes through the steps: (a) observe and assimilate information from the external environment; (b) decide on goals according to your private *goal decision* preference policy; (c) plan and choose plans according to a private *plan selection* preference policy; (d) negotiate for needs (resources) required by plans; (e1) if negotiation is successful then acquire the promised needs (and deliver promised terms); (e2) if negotiation is unsuccessful then choose a new plan and return to (d); (f) execute plans and return to (a).

In a social environment, like e.g. a institution, the negotiation between agents is typically required to conform to some protocol, through which the society requires a certain type of behaviour from the agents. In general, a society protocol can be reflected into a preference policy that is modularly integrated as a subpart of the cycle policy of each agent (for fully compliant agents). Then restricting our attention mainly on the negotiation, the main preferences in the operational behaviour of the agents, captured by their cycle theory, are: (i) prefer to negotiate before starting to execute (e.g. consume resources) your plans – *collaborative/work jointly behaviour* (ii) prefer to change plan when asked to do so by another agent even when your current plan is feasible, – *philagentic behaviour* (iii) prefer to stay with the plan that you had when another agent opened a negotiation dialogue and prefer to continue an open dialogue rather than start a new one – *coherent behaviour* (iv) prefer to acquire needs only after promises have been agreed for these – *consistent behaviour*.

We will consider only a simple example of a society negotiation protocol as depicted in Fig. 1 as a finite state machine. S_1 is an initial state, S_2 is a successful final state and S_3, S_5, and S_6 are unsuccessful final states. A single line arc shows an exchange of a message between the two agents while a double line arc shows a negotiation conversation between the agents, where several message exchanges may occur. The protocol on the right side of Fig. 1 is part of the

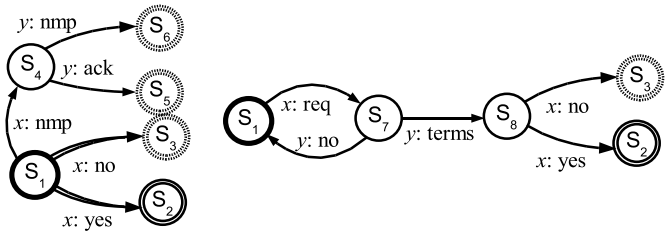

Figure 1. Protocol

protocol synthetically depicted on the left side by double line arcs. Here x, the initiator, asks a set of *Needs* as specified in the initial part P^R of its currently selected plan P in order to make this feasible. At state S_2, the negotiation conversation finishes successfully (by x accepting the terms of y) and hence both current plans of x and y are feasible. At state S_3, the negotiation conversation finishes by x refusing y's terms. Hence x's plan is feasible, but y's is not. Agent y will next choose a (new) plan and start a new dialogue asking x for its needs, following the same protocol with exchanged roles.

Another possibility is, from S_1, to notify that there are no more plans left (*nmp*): thus, at state S_4, agent y deletes its current plan - as x cannot make any of its plans feasible under this - and picks a new plan notifying x of this by an "ack" message (reaching S_5). This thus encodes an extreme cooperative or *philagentic* behaviour, by the agents operating under this protocol. Alternatively, at state S_4, y realizes that its has no more new plans and causes the overall negotiation process to terminate with failure (S_6).

We now present a *cycle of operation* for an *extremely cooperative (or philagentic) behaviour*, of an agent Ag, that is compiled from a cycle theory with preferences given above and that encompasses this society protocol. Here S denotes the set of plans for a given set of goals, initially empty, that the agent has currently deleted from consideration. *OD*(Ag) or *OD*(Ag, Π) denotes the current negotiation dialogue opened by an initial request by the agent Ag where Π denotes the set of plans that Ag has tried so far within this dialogue.

1. Goal Introduction (GI): Decide top-level goals Gs.

2. Plan Introduction (PI): Choose a plan P for Gs s.t. $P \notin S$. If *OD*(Ag, Π), an additional condition is: $P \notin \Pi$.

3. Negotiation Dialogue (ND):

- **3.0 - If receive "fail":** Terminate with failure. Return to step 1.
- **3.1 - If no (new)** P **exists and** ***OD***(Ag)**:** Send "nmp" message, wait for *OD*(Ag) to be closed and return to 3.
- **3.2 - If no (new)** P **exists and not** ***OD***(Ag)**:** Terminate with failure. Send "fail". Return to step 1.
- **3.3 - If received "nmp" and** P **exists:** Delete plan P, i.e. add P to S. Send "ack" message closing *OD*(Ag'). Return to 2.
- **3.4 - If received "nmp" and no** P **exists:** Terminate with failure. Send "fail". Return to step 1.

3.5 - If P exists and *OD*(Ag, Π): Replace Π with $\Pi \cup \{P\}$. Start *Negotiation Conversation NN*(P) for the *Needs* (possibly empty) in the plan P.

3.6 - If P exists and no dialogue is open: *Either* open new dialogue *OD*$(Ag, \{P\})$, and start *Negotiation Conversation NN*(P): for the *Needs* (possibly empty) in the plan P, *or* wait until *OD*(Ag') is opened by $Ag' \neq Ag$, and start *Negotiation Conversation NT*(P): to negotiate with *Terms* equal to the *Needs* (possibly empty) that it has in the plan P.

4.1 - If *NN* or *NT* succeeds Action Execution (AE(*Needs*)): Agent Ag gets its *Needs* and gives the *Terms* that it has promised. Dialogue closes.

4.2 - If *OD*(Ag) and *NN* ends with msg(*from*$(Ag'), no$(*Needs, Terms*))**:** Go to step 2 with *OD*(Ag) open to try another plan.

4.3 - If *OD*(Ag) and *NN* ends with msg(*from*$(Ag), no$(*Needs, Terms*))**:** Close dialogue *OD*(Ag). Return to 3.6.2.

5. Action Execution AE(P): Execute plan P. Return to 1.

In the above negotiation dialogue the agents engage in two types of *negotiation conversations*: *Negotiation for Needs*, *NN* and *Negotiation for Terms*, *NT*. *NN* refers to the conversation of an agent Ag that makes an initial request for *Needs*; *NT* to the reply to such a request, when the agent negotiates for terms in return. They depend on the agents' current plans P and on their private negotiation policies, *KB*$_{Neg}$. In fact, the agents will decide how to proceed in the conversation, based on their *KB*$_{Neg}$ and preference reasoning $\models_{pr}$, as presented in 2.2. They terminate successfully iff either of the agents sends a $msg(to(\textit{Peer}), yes(\textit{Needs}, \textit{Terms}))$ message. *NN* opened by Ag ends in failure either when Ag receives $msg(\textit{from}(Ag'), no(\textit{Needs}))$ or when it sends the message $msg(\textit{from}(Ag), no(\textit{Terms}))$. *NT* fails immediately with *NN*.

We note that depending on the negotiation policy, e.g. cooperative or non-cooperative, we can obtain different behaviours in the these conversations. Thus different ways to use the negotiation protocol are possible by a simple modular change of this private policy.

3 Formal Results

Let $\mathcal{P}_x(Gs)$ be the set of alternative plans, generated by agent x to achieve Gs. We call a *plan* $P \in \mathcal{P}_x(Gs)$ *feasible* if x initially has all the resources to carry it out or if x has been promised those missing (*Needs*). We call a *goal* G of an agent x *feasible* if there exists a feasible plan $P \in \mathcal{P}_x(Gs)$. Finally, we call a $\mathcal{KGP}$*-agent* an agent using the negotiation policies defined in Sec. 2.2 and cycle operations described in Sec. 2.3.

We are now able to state some properties about the system. We will focus on 2-agent situations such that there exists a possible choice of plans for x and y and a reallocation of resources which makes such plans feasible ($solvability$ assumption).

Proposition 1 *Let us consider two $\mathcal{KGP}$-agents, engaged in a negotiation dialogue. Then, the dialogue will comply with the protocol described in Sec. 2.3.*

This result follows by the agent policies and control theory.

Theorem 2 *Given two $\mathcal{KGP}$-agents and a possible exchange of resources that makes their goals feasible ($solvability$ assumption): (i) if both agents are cooperative, there exists a negotiation dialogue achieving such an exchange; (ii) if one agent is cooperative and one is non-cooperative, there exists a negotiation dialogue achieving an exchange that fulfills the plan of the non-cooperative agent.*

This result holds because a cooperative agent is willing to give away a resource for nothing in exchange, and the philagentic agent cycle of operation implements an exhaustive search in the space of solutions of a resource reallocation problem. We do not say anything instead in case the two agents are both non-cooperative: in fact, if a non-cooperative agent does not have any needs, he will reject all incoming requests, and this may result in failing to find a suitable resource allocation for both agents.

Corollary 3 *Given two cooperative $\mathcal{KGP}$-agents, under the $solvability$ assumption, there exists an operation of the two agents that successfully executes their plans.*

In fact, by Theor. 2 there exists a negotiation dialogue which achieves an exchange of resources that makes both agents' goals feasible, and the control of operation described in Sec. 2.3 allows to find a combination of choices of plans which allows for such a dialogue, and to produce the dialogue itself.

Theorem 4 *Given two $\mathcal{KGP}$-agents, under the $solvability$ assumption, then the agents will execute their plans successfully (in an ideal external world).*

This result follows from Cor. 3. In fact, as specified by the cycle of Sect. 2.3, $\mathcal{KGP}$*-agents* do not consume their resources until a dialogue has successfully terminated, thus preventing agents from following a plan which consumes resources needed to the other agent, while they could follow instead some alternative plan which accommodates both agents' needs.

We conclude this section with a remark. While negotiation for terms may not be essential in ideal worlds of collaboration, it is indeed natural in open environments, where requests could be refused e.g. by non-cooperative agents. Therefore, while we focussed on results about agents following a specific behaviour, such as cooperative and philagentic, we provide a framework which is indeed open to heterogeneity, and to more general profiles of agents where their inclination to collaborate could depend on roles and context.

4 Agent problem solving Behaviour

In this section we briefly demonstrate the application of our framework to an example problem proposed in [13]. The example can be summarised as follows:[3] John has the goal to listen to music and Peter the goals of returning his books to the library and have beer. John has \$10, a CD, and a broken CD player. He is able to play music and return books to the library at no cost. Peter has \$15. To get the beer he needs \$25. He is able to return the books, by paying \$10 for the taxi. He is also able to repair broken CD Players (whereas John does not have this capability himself).

Agents are unaware of each other's capabilities. They only know which of their (sub)goals can be requested to other agents. Such requestable goals are: *workingCDP* for John, *books_returned* for Peter, and the resource $money$ for either agent. The individual expertise (capabilities) of each agent is captured simply by adding domain specific knowledge in their respective T_{AEC}. For instance:

KB$_{John}$: $init(play(CD), T, music) \leftarrow$
$holds(use\_of(workingCDP), T), holds(use\_of(CD), T).$
$init(return(Books), T, books\_returned(Books))$

KB$_{Peter}$: $init(return(Books), T, books\_returned(Books)) \leftarrow$
$holds(use\_of(money, 10), T).$
$init(\textit{repairCDP}, T, \textit{working\_CDP}).$

Here $money$ is a cumulative resource and in $use\_of(money, Q)$ the number, Q, indicates its quantity.

The agents start their operation (asynchronously), following their cooperative cycle of operation as in section 2.3, by deciding, according to their goal decision policy, their top-level goals of: $music$ for

[3] In [4] this example is worked out in detail.

John and $beer, books\_returned$ for Peter. They then generate the following conditional plans for these goals, where again we have assumed that their plan selection preference policy picks these plans given the current conditions:

$P_{John} = \{request(\textit{workingCDP}, T_0),\ get(\textit{workingCDP}, T_1)\}$
$\cup \{use(my\_cd, T_2),\ use(\textit{workingCDP}, T_2),\ play(my\_cd, T_3)\}$

$P_{Peter} = \{request((money, 20), T'_0),\ get((money, 20), T'_1)\}$
$\cup \{use((money, 10), T'_2), return(my\_books, T'_2),$
$use((money, 25), T'_3),\ buyBeer(T'_3)\}$

which are conditional on $holds(promise(\textit{workingCDP}), T_1)$ and $holds(promise((money, 20)), T'_1)$, respectively, where $T_0 < T_1 < T_2 < T_3, T'_0 < T'_1 < T'_2$ and $T'_1 < T'_3$.

At this stage one of the agents starts a negotiation dialogue for the needs in its plan. Suppose that John executes the action $tell(peter, msg(\textit{from}(john), req(\textit{workingCDP}))$, requesting the need *workingCDP*. Peter accepts, asking for $\textit{Terms} = \{(money, 20)\}$ which in turn John refuses as it does not have them available. John has no other plan and thus it notifies Peter via a "nmp" message. Peter, conforming to the cooperative protocol, deletes its plan, and finds a new plan which explicitly requests $books\_returned$, along with $10 for the goal of $beer$. They will then continue the negotiation and irrespective of who makes the first request they will now agree, as their needs can be satisfied by each other. Once the agents then have promises from each other for their *Needs*, they can proceed to acquiring them (get action) and finally executing the remainder of their plans to satisfy their goals.

5 Discussion

In this paper, we presented a logic-based framework that synthesizes in a modular way three aspects of agency: planning, negotiation, and control of operation. The main advantages of our approach are in the use of a declarative formalism, in its modularity and openness to extension, and in its computational realizability.

In [1], Amgoud et al. present a framework where argumentation is used to support agent dialogues. The authors focus on the use of argumentation for agent negotiation, rather than on the overall operational agent framework. Building on [1], Sadri et al. [15] propose an operational framework for automated negotiation processes based on abductive logic programming, and start investigating some properties of the framework. In the cited work and its following extensions, the authors focus on the ability of agents to produce dialogues, on protocol compliance, and on ability to solve resource reallocation problems, achieving results which are comparable with ours. However, their architecture, focussing on the reasoning needed to produce dialogues, is independent of other agent capabilities, such as planning. Moreover, differently from [15], our use of preference reasoning allows for a modular agent programming.

IMPACT is an agent platform mainly developed at the University of Maryland [2], where the authors use a deontic formalism to guide the agent deliberation process, by calculating a "deontically consistent" stable model [5]. Despite the common formal ground, based on logic programming, our approaches are very different: IMPACT agents can be seen as wrappers that can "agentify" other components such as heterogenous data sources and planners. In this work instead we present a modular though unified architecture, aimed at programming agents in a flexible and declarative way, and at being able to prove properties about their behaviour.

Finally, the example presented in Sect. 4 was first proposed by Küngas and Matskin [13], to show agent cooperative problem solving achieved by use of linear logic and partial deduction. Our work differs from [13] in several aspects, including *autonomy*, as the amount of information shared by agents is tuned by the policies, and agents are not necessarily cooperative, *planning capabilities*, *modularity* of knowledge representation, and finally the ability to reason on dynamic and contextual information.

Our framework is able to accommodate several variations and extensions of agent distributed problem solving. These include problems where the agents are heterogeneous, e.g., where some agents are cooperative and some are not. Similarly, we can accommodate issues that relate to adapting the negotiation to different circumstances, e.g., the type of requested need or the relative roles of the agents involved, by exploiting the added flexibility of the private agent policies. We conclude by discussing some weak points of our approach so far. We made the simplifying assumptions that agents negotiate for all their needs together and in advance. In more realistic scenarios, agents might need to interleave plan execution with the negotiation of needs (which still can be dealt with by conditional planning). Also, in general there are more than two agents involved in a resource reallocation scenario. In the future, besides addressing these technical limitations, we would like to use our prototypical implementation to perform an extensive testing on a number of scenarios, also to determine to which extent this negotiation methodology scales up.

ACKNOWLEDGEMENTS

This work was supported by the European Commission FET Global Computing Initiative within the SOCS project (IST-2001-32530), and by the MIUR COFIN 2003 project *Sviluppo e verifica di sistemi multiagente basati sulla logica*.

REFERENCES

[1] L. Amgoud, S. Parsons, and N. Maudet, 'Arguments, dialogue and negotiation', in *Proc. ECAI*, IOS Press, (2000).
[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, (1999).
[3] G. Brewka, 'Well founded semantics for extended logic programs with dynamic preferences', *JAIR*, **4**, 19–36, (1996).
[4] N. Demetriou, A. C. Kakas, and P. Torroni. Further examples of the functioning of computees, Discussion Note, SOCS Consortium, (2003).
[5] T. Eiter, V. S. Subrahmanian, and T. J. Rodgers, 'Heterogeneous active agents, I: Semantics', *AIJ*, **108**(1/2), 179-255, (2000).
[6] K. Eshghi, 'Abductive planning with the event calculus', in *Proc. ICLP*. MIT Press, (1988).
[7] A. C. Kakas, R. A. Kowalski, and F. Toni, 'Abductive Logic Programming', *Journal of Logic and Computation*, **2**(6), 719–770, (1993).
[8] A. C. Kakas, P. Mancarella, and P. M. Dung, 'The acceptability semantics for logic programs', in *Proc. ICLP*, pp. 504–519, (1994).
[9] A. C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni, 'The KGP Model of Agency', in this volume.
[10] A. C. Kakas and P. Moraïtis, 'Argumentation based decision making for autonomous agents', in *Proc.AAMAS*, pp. 883–890. ACM, (2003).
[11] A. C. Kakas and F. Sadri, *Computational Logic: Logic Programming and Beyond*, *LNAI* **2407** and **2408**, Springer, (2002).
[12] R. A. Kowalski, 'Problems and promises of computational logic', in *Proc. Symp. on Comp. Logic*, pp. 1–36. Springer, (1990).
[13] P. Küngas and M. Matskin, 'Linear logic, partial deduction and cooperative problem solving', in *Proc. DALT*, *LNAI* **2990**, Springer, (2004).
[14] A. S. Rao and M. P. Georgeff, 'Modeling rational agents within a BDI-architecture', in *Proc. (KR&R)*, pp. 473–484. MKP, (1991).
[15] F. Sadri, F. Toni, and P. Torroni, 'Logic agents, dialogues and negotiation: an abductive approach', in *Proc. AISB*, (2001).
[16] M. Shanahan, 'Solving the frame problem: a mathematical investigation of the common sense law of inertia'. MIT Press, (1997).
[17] K. Stathis, A.C. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. 'PROSOCS: a platform for programming software agents in computational logic.' In *Proc. AT2AI-4/EMCSR Session M*, (2004).