Using the DTC Scheduler to Improve Intention Selection in BDI Agents

Rafael H. Bordini[1], Ana L. C. Bazzan[1],
Rafael de O. Jannone[1], Daniel M. Basso[1],
Rosa M. Vicari[1], and Victor R. Lesser[2]

[1] Informatics Institute
Federal University of Rio Grande do Sul
Av. Bento Gonçalves 9500, CP 15064
91501-970, Porto Alegre-RS, Brazil
{bordini,bazzan,jannone,dmbasso,rosa}@inf.ufrgs.br

[2] Department of Computer Science
University of Massachusetts
Amherst, MA 01003
lesser@cs.umass.edu

Abstract

This paper reports on the developments with the project entitled "Combining Cognitive and Utilitarian Coordination in a Layered Agent Architecture", supported by CNPq/ProTeM-CC and the NSF. It focuses on an initial idea towards that goal, which is to use the Design-To-Criteria (DTC) scheduler in order to produce an efficient intention selection function for AgentSpeak(XL), an extension we have proposed to a known BDI agent oriented programming language. In this paper, we concentrate on the particular extensions to the language and its interpreter that were necessary so that the integration with DTC was possible. This includes the way in which an agent's set of intentions (in the extended version of the interpreter) is translated into a TÆMS task structure, in order for DTC to schedule those intended plans. We also mention some future work on alternative ways to handle that integration and the assessment of this unified framework.

1 Introduction

In the course of attempting to develop a powerful and innovative agent framework based on a two-level agent architecture combining cognitive and decision-theoretic frameworks, several ideas have occurred to the research groups involved in this project which only partially achieve that goal. This paper presents one of those ideas which, although not fully integrating a Beliefs-Desires-Intentions (BDI) [Rao and Georgeff, 1995]

framework with the Generalized Partial Global Planning (GPGP) coordination mechanisms [Decker and Lesser, 1995] in a two-level agent architecture, it significantly contributes to the performance of the BDI framework we have been developing [Machado and Bordini, 2001], based on Rao's AgentSpeak(L) BDI agent oriented programming language [Rao, 1996a].

The referred idea is to use the DTC scheduler [Wagner et al., 1998], which is part of the GPGP framework, to improve the performance in interpreting BDI programming languages, in particular concerning issues such as intention selection, on which we concentrate in this paper. In fact, this idea is being applied to two different BDI frameworks; we are using a variation of AgentSpeak(L) and another group within the Brazilian team is using the X-BDI framework [Móra et al., 1999]. This paper reports specifically the experiences with the former. The latter is being considered in [Móra and Vicari, 2001].

Pursuing this more quickly achievable goal has helped clarifying several aspects of the technologies we have at hands and pointing to future directions on how to pursue our more ambitious project goals.

This paper is structured as follows. The next section provides the necessary background on AgentSpeak(L) and GPGP. Section 3 presents the original ideas underlying this project, whereas Section 4 presents the specific ideas we are pursuing at the moment. It presents AgentSpeak(XL), an extension we have proposed to AgentSpeak(L), both for improving that language in various aspects and in particular to accommodate the on-the-fly use of DTC for generating an efficient intention selection function.

2 Background

2.1 AgentSpeak(L) and the SIM_Speak Interpreter

In [Rao, 1996a], Rao introduced the AgentSpeak(L) programming language. In that paper, not only has he defined formally the operation of an abstract interpreter for it, but he has also defined a proof theory for that language in which, he claimed, known properties that are satisfied by BDI systems using BDI Logics [Rao, 1996b, Singh et al., 1999] could also be proved; further, he claimed that there is an one-to-one correspondence between his interpreter and the proof system. In this way, he proposed what can be considered as the first viable approach to bridging the ever so quoted gap between BDI theory and practice: an old worry of the BDI community.

Also, AgentSpeak(L) has many similarities with traditional logic programming, which is another characteristic that would favour it to become a popular language: it should prove quite intuitive for those familiar with logic programming. Besides, it has a neat notation and provides quite elegant specifications of BDI agents. This could contribute for its being considered also for the production of formal, high-level specification of BDI systems.

Further formalisation of the AgentSpeak(L) abstract interpreter and missing details were given in [d'Inverno and Luck, 1998] using the Z formal specification language; most of the elements in that formalisation had already appeared in [d'Inverno et al., 1998]. This highlights the fact that AgentSpeak(L) is strongly based on the experience with Kinny's dMARS [Kinny, 1993].

However, until recently there was no implemented interpreter for AgentSpeak(L). We have presented the very first working interpreter for AgentSpeak(L) in [Machado and Bordini, 2001], which we called SIM_Speak. This was a prototype interpreter based on Sloman's SIM_AGENT toolkit [Sloman and Poli, 1996]. For this project, we are developing an efficient interpreter in C++, and we extended the language so as to integrate with DTC (see Section 4).

We are presently using SIM_Speak in practical applications of multi-agent systems. In [Rossetti et al., 2001], we proposed a new approach to traffic simulation, where drivers are endowed with cognitive decision-making capabilities. Human factors such as personality and its influence on route choice are not accounted for in traditional traffic simulation. In the paper, we provided a reasoning mechanism for drivers, which were modelled as intentional entities through the use of a BDI architecture. We used AgentSpeak(L) to specify such agents, which decide on particular departure times and routes to take, based on their specific goals for each simulated day and according to their particular past experiences with the routes they know. In that way, that paper contributed to showing a practical way to represent and assess driver behaviour, and the adequacy of using AgentSpeak(L) as a modelling language, as it provides clear and elegant specifications of BDI agents. Those specifications are presently being used for running simulations on SIM_Speak.

The next section gives an overview of the AgentSpeak(L) programming language and its abstract interpreter, as presented in [Machado and Bordini, 2001].

2.1.1 AgentSpeak(L) Syntax and Informal Semantics

This section covers the basics of the syntax and informal semantics of AgentSpeak(L) first given in [Rao, 1996a], the paper which introduced it. Although in this section the formal definitions of the AgentSpeak(L) syntax are reproduced almost like the ones given in that paper (except that we have improved some of them here), this presentation is indispensable, together with the presentation of SIM_AGENT in the next section, for the understanding of the reminder of the paper. As we mentioned before, besides [Rao, 1996a], the reader can refer to [d'Inverno and Luck, 1998] for detailed formalisation of the language.

An AgentSpeak(L) agent is created by the specification of a set of base beliefs and a set of plans. The definitions below introduce the necessary notions for the specifications of such sets. Those familiar with Prolog, when reading actual examples of AgentSpeak(L) programs (e.g. in [Rao, 1996a] and [Rossetti et al., 2001]) will notice many similarities, including the convention of using uppercase initials for variable identifiers, and the issues related to predicates and unification.

Definition 1 *If b is a predicate symbol, and $t_1, \ldots, t_n$ are terms then $b(t_1, \ldots, t_n)$ or $b(\mathbf{t})$ is a* belief atom. *A ground belief atom is a* base belief. *A belief atom $b(\mathbf{t})$ or its negation $\neg b(\mathbf{t})$ are* belief literals. *Belief literals are* beliefs. *If φ and ψ are beliefs, $\varphi \wedge \psi$ is also a belief.*

AgentSpeak(L) distinguishes between two types of goals: *achievement goals* and *test goals*. Achievement goals are predicates (as defined for beliefs above) prefixed with the '!' operator, while test goals are prefixed with the '?'operator. Achievement goals state that the agent wants to achieve a state of the world where the associated predicate is true.

Test goals state that agents wants to test whether the associated predicate is a true belief (i.e., whether it can be unified with that agent's base beliefs).

Definition 2 *If g is a predicate symbol, and $t_1, \ldots, t_n$ are terms then $!g(t_1, \ldots, t_n)$ or $!g(\mathbf{t})$ and $?g(t_1, \ldots, t_n)$ or $?g(\mathbf{t})$ are* goals.

Next, the notion of triggering event is introduced. It is a very important concept in this language, as it is a triggering event that starts off the execution of plans. There are two types of triggering events: those related to the *addition* ('$+$') and those related to the *deletion* ('$-$') of mental attitudes (beliefs or goals, in fact).

Definition 3 *If $b(\mathbf{t})$ is a belief atom, $!g(\mathbf{t})$ and $?g(\mathbf{t})$ are goals, then $+b(\mathbf{t})$, $-b(\mathbf{t})$, $+!g(\mathbf{t})$, $-!g(\mathbf{t})$, $+?g(\mathbf{t})$, and $-?g(\mathbf{t})$, are* triggering events.

Clearly, from the usual model of agents (see, e.g., the diagram used in [Wooldridge, 2000]), in regard to their acting on a environment, one sees that plans need to refer to the basic actions that an agent is able to perform on its environment. Such actions are defined below.

Definition 4 *If a is an action symbol and $t_1, \ldots, t_n$ are first-order terms, then $a(t_1, \ldots, t_n)$ or $a(\mathbf{t})$ is an* action.

We now round off the presentation of AgentSpeak(L) syntax with the definition of plans. Recall that the designer of an agent using AgentSpeak(L) does so by specifying a set of base beliefs and a set of plans only. An AgentSpeak(L) plan has a head which is formed of a triggering event (the purpose for that plan), and a conjunction of belief literals forming a context that needs to be satisfied if the plan is to be executed (the context must be a logical consequence of that agent's set of base beliefs). A plan has also a body, which is a sequence of basic actions or (sub) goals that the agent has to achieve (or test).

Definition 5 *If e is a triggering event, $b_1, \ldots, b_m$ are belief literals, and $h_1, \ldots, h_n$ are goals or actions, then $e : b_1 \wedge \ldots \wedge b_m \leftarrow h_1; \ldots; h_n$ is a* plan. *The expression to the left of the arrow is referred to as the* head *of the plan and the expression to the right of the arrow is referred to as the* body *of the plan. The expression to the right of the colon in the head of a plan is referred to as the* context. *For convenience, we shall rewrite an empty body with the expression "$true$".*

We now turn to providing the basics on the interpretation of AgentSpeak(L) programs. For that, we need a few more definitions.

Intentions are particular courses of actions (in the form of a stack of partially instantiated plans) to which an agent has committed in order to achieve a particular goal, and are represented as defined below.

Definition 6 *Each* intention *is as stack of* partially instantiated plans, *i.e., plans where some of the variables have been instantiated. An intention is denoted by $[p_1 \ddagger \ldots \ddagger p_z]$, where p_1 is the bottom of the stack and p_z is the top of the stack. The elements of the stack are delimited by $\ddagger$. For convenience, we shall refer to the intention* `[+!true:true <- true]` *as the* true intention.

Events, which may start off the execution of plans, can be external, when originating from perception of the agent's environment, or internal, when generated from the agent's own execution of a plan (e.g., a subgoal in a plan is an addition of goal which may be a triggering event). In the latter case, the event is accompanied by the intention which generated it.

Definition 7 *Each* event *is a tuple* $\langle e, i \rangle$*, where* e *is a triggering event and* i *is an intention. If the intention* i *is the* true intention*, the event is called an* external event*; otherwise it is an internal event.*

We can now formally define an AgentSpeak(L) agent, as follows.

Definition 8 *An* agent *is given by a tuple* $\langle E, B, P, I, A, \mathcal{S}_\mathcal{E}, \mathcal{S}_\mathcal{O}, \mathcal{S}_\mathcal{I} \rangle$*, where* E *is a set of events,* B *is a set of base beliefs,* P *is a set of plans,* I *is a set of intentions, and* A *is a set of actions. The selection function* $\mathcal{S}_\mathcal{E}$ *selects an event from the set* E*; the selection function* $\mathcal{S}_\mathcal{O}$ *selects an option or an applicable plan from a set of applicable plans; and* $\mathcal{S}_\mathcal{I}$ *selects an intention from the set* I*.*

We have devised a diagram which explains informally the functioning of an interpreter for AgentSpeak(L) (note that this is formalised in [Rao, 1996a] and [d'Inverno and Luck, 1998]). The pictorial description of such interpreter, given in Figure 1, greatly facilitates the understanding of the interpreter for AgentSpeak(L) proposed by Rao. In the figure, sets (of beliefs, events, plans, and intentions) are represented as rectangles. Diamonds represent selection (of one element from a set). Circles represent some of the processing involved in the interpretation of AgentSpeak(L) programs.

At every cycle of execution of an agent program, AgentSpeak(L) updates a list of events, which may be generated from perception of the environment, or from the execution of intentions (when subgoals are specified in the body of plan). Note that we have introduced a Belief Revision Function (BRF) in the architecture which is implicit in Rao's interpreter (but normally made explicit in the generic BDI architecture [Wooldridge, 1999]). It is assumed that beliefs are updated from perception and whenever there are changes in the agent's beliefs, this implies the addition of an event in the set of events. We have made this process explicit in the figure, by including the BRF component.

It is important to remember that $\mathcal{S}_\mathcal{E}$, $\mathcal{S}_\mathcal{O}$, and $\mathcal{S}_\mathcal{I}$ (see Definition 8) are part of the definition of an agent. Neither Rao nor d'Inverno and Luck elaborate on how users specify such functions, but they are assumed to be agent-specific. After $\mathcal{S}_\mathcal{E}$ has selected an event, AgentSpeak(L) has to unify that event with triggering events in the heads of plans. This generate a set of all *relevant plans*. When unifying the context part of heads of plans in that set with the agent's base beliefs, AgentSpeak(L) determines a set of *applicable plans* (plans that can actually be used for the event to be handled). Then $\mathcal{S}_\mathcal{O}$ chooses a single applicable plan from that set, and either pushes that plan on the top of an existing intention (if the event was an internal one), or creates a new intention in the set of intentions (if the event was external, i.e., generated from perception of the environment). An intention is, therefore, a stack of partially instantiated plans.

All that remains to be done at this stage is to select a single intention to be executed in that cycle. Note that each external event for which there is an applicable plan generates

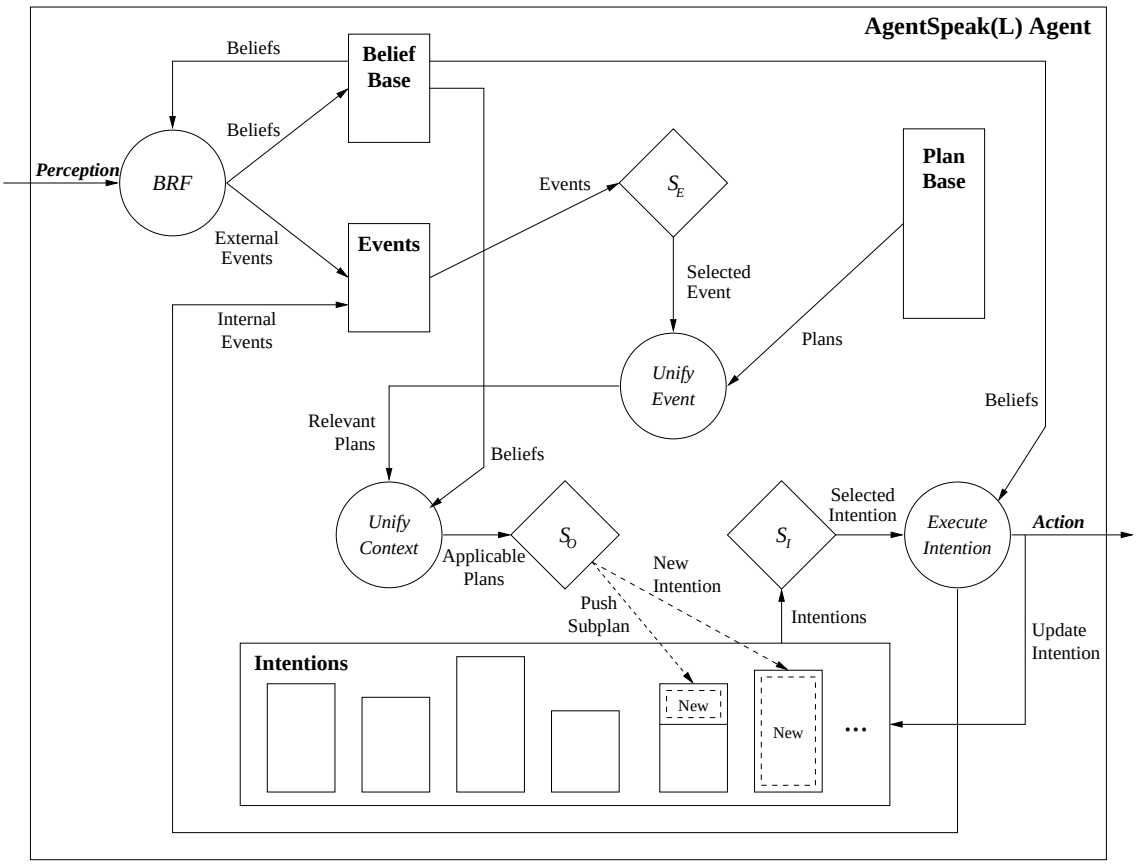

Figure 1: Interpreting AgentSpeak(L) Programs [Machado and Bordini, 2001]

an independent stack of partially instantiated plans within the set of intentions. The $\mathcal{S}_{\mathcal{I}}$ function selects one intention from the ones which stay on the top of all those independent stacks within the set of intentions. This intention is then executed, which implies that either a basic action is performed by the agent on its environment, an internal event is generated (in case the subgoal is an achievement goal), or a test goal is performed (which means that the set of beliefs need to be consulted). If the intention is a basic action or a test goal, the set of intention needs to be updated. In the case of test goals, further variable instantiation will occur in the partially instantiated plan which contained that test goal (and the test goal itself is popped from the stack). In the case where a basic action is selected, the necessary updating of the set of intentions is to pop that action from the intention stack. When a popped intention marks the end of a subplan, the subgoal that generated it (which therefore stays immediately below the previously pushed subplan within the stack) is also popped from the stack, or the whole intention is removed from the set if the initial plan (i.e., the plan triggered by an external event) is the one that finished execution. This ends a cycle of execution, and AgentSpeak(L) starts all over again, checking the state of the environment after agents have acted on it, generating events, and so forth.

2.2 GPGP, TÆMS, and DTC

Generalized Partial Global Planning (GPGP) has its roots in Durfee's Partial Global Planning (PGP) work on the Distributed Vehicle Monitoring Testbed (DVMT) [Durfee and Lesser, 1991], and von Martial's work on recognizing and taking advantage of a "favors" relationship [von Martial, 1992]. It is one of the best known coordination mechanisms in the MAS research community, and sprang out of Prof. Victor Lesser's research group. The key insights that led to the GPGP approach were: (1) that coordination activities like communicating meta-information, recognizing relationships, and updating scheduling criteria could be separated from the actual process of local scheduling; and (2) that specific relationships that showed important in the DVMT domain actually had generic counterparts that could appear in any domain. The generic task relationships identified include both hard relationships such as the results from one task enabling another task, and soft relationships such as facilitation.

GPGP was developed as a domain-independent framework for coordinating real-time activities of small teams of agents [Decker, 1996, Decker and Lesser, 1995]. This framework supports a set of coordination mechanisms which can recognize the features of the agent's subjective view of a given problem-solving process, trigger information gathering actions when needed, and handle redundancy as well as soft and hard relationships appropriately. The gathered information is used by agents to commit themselves to the execution of certain methods at given deadlines and characteristics (e.g. a given quality level). Over the last three years, the framework has been extended based on the experience acquired when applying it to a number of applications which included: distributed situation assessment, information gathering and management, coordination of concurrent engineering activities, and hospital scheduling [Decker, 1998, Decker and Li, 1998, Nagendra Prasad and Lesser, 1996, Oates et al., 1997]. This led to the creation of the GPGP2 framework [Lesser et al., 1998, Lesser, 1998, Lesser et al., 1999, Vincent et al., 2001]. Next, we make a brief discussion of GPGP and how it permits the implementation of coordination strategies for a wide range of multi-agent systems.

The basic idea behind GPGP is that each agent constructs its own local view of the activities (that it intends to pursue in the short-to-medium-term time frame), and the relationships among these activities. This view can be augmented by information from other agents, thus becoming a view that is not entirely local. Individual coordination mechanisms that are part of GPGP help to construct these partial views, and to recognize and respond to particular task structure relationships by making commitments to other agents. These commitments result in a more coherent, coordinated behavior by affecting the tasks an agent will execute, when they will be executed, and whom their results will be transmitted to. The set of mechanisms are: (1) communicate non-local views; (2) communicate appropriate results; (3) avoid redundancy; and (4,5) handle hard and soft coordination relationships which exist between tasks of two different agents.

GPGP uses the TÆMS (Task Analysis, Environment Modeling, and Simulation) domain-independent framework to represent formally the coordination aspects of problems. Unlike many systems, this framework deals with worth-oriented [Rosenschein and Zlotkin, 1994] environments where a goal is not either fully achievable or not at all, but rather has a degree of achievement associated with it. In TÆMS,

this is represented by tracking a quantitative vector of task characteristics or criteria over which some utility preference may be expressed [Wellman and Doyle, 1992]. We also allow many task structures to be active at once, representing several objectives all of which must be achieved to some degree. The agent's task structure view may change over time due to uncertainty, or due to a dynamically changing environment. TÆMS also provides ways to model scenarios where tasks have deadlines and particular kinds of results must be achieved. In that case, we say that a certain task's quality has been accrued.

The central representation in TÆMS, which drives all of the GPGP coordination mechanisms is that of the local (and non-local) sets of activities called task structures. Several important pieces of information are captured in the task structure. These include: (a) the top-level goals/objectives/abstract-tasks that an agent intends to achieve; (b) one or more of the possible ways that they could be achieved, expressed as an abstraction hierarchy whose leafs are basic action instantiations, called methods; (c) a precise, quantitative definition of the degree of achievement in terms of measurable characteristics such as solution quality, cost, and time; (d) task relationships that indicate how basic actions or abstract task achievement affect task characteristics elsewhere in the task structure. The quality of a task group depends on what and when its subtasks and their methods are executed. Beside the local effects of the execution of methods on the quality and duration of their supertasks, there exist non-local effects (NLE) like enables, facilitates, and so on. A task T may enable a method M in the sense that the quality of M cannot be accrued until T is completed, i.e. the earliest start time of M is the finish time of T. Therefore, enables is a hard relationship, which means it must be observed in all cases. When a task T1 facilitates another task T2, the duration and/or quality of T2 is affected, but may not necessarily be observed since facilitates is a soft relationship. Another important set of NLE applies to methods and their use of resources. Uses, replenishes, etc. are NLE's running from one method to one resource and altering its state. Conversely, limited, exclusive access, etc. are NLE's running from one resource to a method.

Besides using TÆMS as a representation language, GPGP includes also the Design-To-Criteria Scheduler (DTC). It uses a domain-independent, real-time, flexible computation approach to task scheduling. DTC efficiently reasons about the quality, cost, duration of interrelated methods, and constructs a set of satisfying schedules for achieving high-level goals.

The GPGP approach uses the basic TÆMS task structure representation and adds two important extensions: partial representations of the task structures in other agents, and local and non-local commitments to task achievement. Each GPGP coordination mechanism is specified with respect to features in these dynamically changing task structures, and so provides flexible coordinated responses across different problem domains.

3 Project Overview

The CUCLA project (Combining Cognitive and Utilitarian Coordination in a Layered Agent Architecture) is financed by CNPq's ProTeM-CC initiative and the National Science Foundation. It is being developed by a cooperation between the multi-agent systems research groups based at the Federal University of Rio Grande do Sul (Brazil) and at the University of Massachusetts at Amherst (United States of America). The Brazilian group

is coordinated by Rosa Vicari, while the group based in the US is coordinated by Victor Lesser.

CUCLA aims at augmenting an agent language based on mentalistic metaphors with a decision-theoretic approach to agent coordination, thus creating a powerful, innovative multi-agent framework. A two-level agent architecture is to be developed that combines concurrently operating subsystems; at the higher level is a subsystem that reasons about symbolic information – an extended version of a BDI architecture, while a lower level is a subsystem that reasons about quantitative information – an extended version of the GPGP (Generic Partial Global Planning) architecture. In this view, the lower layer pertains to feasibility and implementation operations, i.e., the detailed analysis of candidate tasks and actions, the formation of detailed temporal/resource-specific commitments between agents, and the balancing of non-local and local problem-solving activities. In contrast, the upper layer pertains to domain-specific coordination tasks such as the formation of high-level goals and objectives for the agent, and decisions about whether or not to coordinate with other agents to achieve particular goals or to bring about particular objectives. Detailed domain knowledge is used at this level to make these high-level coordination decisions. In contrast, decisions at the lower level do not need to reason about this detailed domain state. However, reasoning about detailed models of the performance characteristics of activities, such as their temporal scope, quality, and effects of resource usage on performance is necessary at this level.

In this view, the layers are interdependent activities that operate asynchronously. The upper layer provides the performance criteria and the set of candidate tasks for consideration by the lower layer; whereas the lower layer, as the result of its deliberation (in constructing the detailed commitments and temporal sequencing necessary to implement the desired objectives) may provide feedback to the upper layer about the feasibility of certain tasks. In response to this feedback, the upper layer may alter the set of candidate tasks or the performance objectives. Many theoretical questions on each level, as well as in the integration of these levels, should also contribute from the research to be carried out in this international cooperation of research teams.

The development of this two-level agent architecture, and a programming language based on it, will result in making it significantly easier to develop multi-agent systems that can function effectively in open environments such as the Internet where the set of goals to achieve, the set of resources available, and the set of other agents are dynamically changing. In fact, such framework could be used in the development of complex applications of any kind. Multi-agent systems have nowadays marked its presence in the development of highly successful systems in areas which had been considered too complex for computational management. This is the reason for its present popularity. The range of application for MAS techniques are really quite wide: from major industrial applications to the simulation of organizations and communities.

Using this resulting programming language we also aim at developing two application scenarios of interest to the research teams involved in this project. The main objective for such development, inasmuch as the project is concerned, is the evaluation of the joint cognitive and decision-theory framework to be developed during the first part of the project. One of the application scenarios is on Intelligent Homes, an area of application for which the U.S. research teams has marked international recognition. The other application area

is Distance Learning, similarly an area for which the Brazilian research team has a long standing recognition. Besides the use in the evaluation of the project results, these application areas also have, respectively, a huge potential in the production of industrial products and in social terms (regarding access to education). Also, we expect to test the prototype initially in scenarios related to the Internet, in which interesting issues arise, e.g., similar or conflicting intentions of agents leading to redundant tasks.

With the specification and construction of the system prototype we expect the coordination module of GPGP to be capable of dealing with commitments created by an analysis of team work, integration of efforts to solve a given task, and of the multiple plans available. This would mean an improvement in the GPGP capacity of evaluation of plans of actions, since currently only the environment-centred options are considered. Also, the mechanisms we expect to provide would not only be used in our joint framework, but also be of interest for BDI-based research generally, as it represents further work towards a current limitation of these formalisms, namely the incapability of practically dealing with quantitative choices.

In summary, this section has outlined possible ways towards the development of a joint framework based on BDI (beliefs, desires, intentions) techniques and the GPGP framework. The BDI level can establish commitments based on knowledge about the states of the agents (e.g. their beliefs, intentions), letting the quantitative aspects (cost-benefit of actions when resource or other types of constraints are considered) for the GPGP framework and its coordination mechanisms, which proved to be efficient in many scenarios. This joint work has the benefit of dealing with agents' plans (created on its turn by agents' mental states), while the actual actions are evaluated under a more quantitative analysis. This way, the work tackles a drawback of pure intentionally-motivated formalisms and agent architectures, namely their weakness to perform quantitative analysis and thus make choices efficiently, given resources and other constraints which appear in dynamic scenarios.

In the remainder of the paper, we concentrate on a simplified problem, which is however akin to the philosophy of the project, and can produce significant improvements to a BDI agent oriented programming language. It does not fulfill all the ambitions of this project, but significantly contributes to the area and has been helpful in providing us with a deeper knowledge of the technologies we have at hands, and should give further insights into how to fully integrate the frameworks produced by the two cooperating teams.

4 Integrating AgentSpeak(XL) and the DTC Scheduler

This section outlines the major changes that we have proposed to the AgentSpeak(L) language and its abstract interpreter, both for adding general programming features, and in particular for accommodating DTC as a on-the-fly generator of schedules for intended means (i.e., plans), so that an efficient intention selection function can be used as part of the interpreter. This extended version of the language an its interpreter was called AgentSpeak(XL).

4.1 Language Extensions

We have been working on some extensions to AgentSpeak(L) to improve various deficiencies of the language for practical programming. These include, for example, the absence of basic arithmetic and relational operators. Other deficiencies are more significant in the conceptual level of multi-agent systems. One serious disadvantage of AgentSpeak(L) in comparison to other abstract agent oriented programming languages, e.g. 3APL [Hindriks et al., 1999], is that it does not provide ways for dealing with plan failure. In fact, Rao pointed out to AgentSpeak(L) events for goal deletion, syntactically represented as $-!g(\mathbf{t})$ and $-?g(\mathbf{t})$, which supposingly were intended for dealing with plan failures, but he did not include that in the semantics of the language.

We have defined a precise mechanism for allowing programmers to use such events in order to handle plan failures. Another significant improvement that we have done is in respect to agent communication. We have devised the means by which AgentSpeak(XL) agents can communicate using KQML [Mayfield et al., 1996], and we have defined also the changes in the abstract interpreter that have to be made for agents' mental state to reflect the communications in which they engage. We emphasise that, although these extensions are well defined, the formal semantics of these extensions is not as yet specified. We do not give even informal accounts of these extensions here, as the main focus of this paper is the integration with DTC.

For the integration with DTC, we need to specify labels which unequivocally identify each individual plan in the plan library. Plan labels are separated from the rest of the plan's syntactic structure by a "`->`" symbol. This was the first syntactic change we introduced in AgentSpeak(XL). A plan is now defined as:

$$label \texttt{ -> } event(E) \texttt{ : } b_1(T_1) \texttt{ \& } \ldots \texttt{ \& } b_n(T_n) \texttt{ <- } a_1 \texttt{ ; } \ldots \texttt{ ; } a_n$$

In order to allow AgentSpeak(XL) programmers to use fundamental constructs in any programming language[1], such as basic arithmetic and relational operators, we have extended the language with a construct we call *internal actions*. They allow the access to user-defined, extensible libraries of procedures, which unlike agent's basic actions (here also referred to as *external actions*), do not affect the environment shared by all agents in a society. The fact that they do not affect the environment is an essential part in the semantics of internal actions: this means that they can be instantaneously executed (as they cause no effects in the environment, and therefore in perception). Because these actions are executed instantaneously, unlike external actions which require the interpreter to proceed to a next cycle (performing the action in the environment and then providing the agent with new perceptions), it means that we can effectively use them in the context part of plans, as well as in the body. Recall that the context part of a plan has to be fully evaluated when the interpreter is checking for applicable plans for a specific event (this cannot way for another interpretation cycle). Also, the possibility of using internal actions in the context of plans is quite important, as programmers may quite often need to have access to those library procedures for actually deciding on whether a plan is applicable or

[1]It is important to note, however, that AgentSpeak(L) is not intended as a general programming language, but for the specification of the high-level reasoning of (BDI) agents. Nevertheless, certain basic programming constructs are quite helpful in that task, according to our experience in programming with AgentSpeak(L), e.g. in [Rossetti et al., 2001].

not. For example, an internal action which implements relational operators may need to be used in the context part of the plan to make sure that it will not even be considered as applicable (executing that action in the body of the plan would first allow the plan to be applicable, possibly chosen as intended means, and only later failing; in that case it would have to be treated with plan failure operators).

Syntactically, internal actions have a '.' in their names, which is used to separate the name of the library from the name of the action (as with C++ classes and methods). This has two advantages: (i) the interpreter can differentiate formulæ in the context of a plan which are predicates that need to be checked for logical consequence of the belief base from internal actions (recall that their semantics assures that they can be performed in that same interpretation cycle), or differentiate, in the body of plans, internal from basic actions (which are dealt with by the environment and whose effects are only perceived in the next interpretation cycle); and (ii) programmers can organise newly defined actions in various libraries.

A *standard library* is provided with AgentSpeak(XL), which define useful operators (e.g., relational and arithmetic ones). This is the only "nameless" library, so one can use $.gte(X, Y)$ to access the gte internal action defined in the standard library. Using something like "... & $(X >= Y)$ & ..." in a plan context is automatically translated into "... & $.gte(X, Y)$ & ...", which in turn makes access to the standard library.

This definition of internal actions has also been quite handy in the aimed integration with DTC. Besides the standard one, another library is provided by AgentSpeak(XL), which is called the task structure library. This is the extension of main interest in this paper, as it allows the use of DTC in intention selection. This library is presented in Section 4.3, as we need first to define the changes we had to introduce in the AgentSpeak(L) interpreter so as to be able to use DTC.

4.2 Extensions to the Interpreter

This section mentions exclusively the changes we propose to the AgentSpeak(L) interpreter in order to integrate with DTC. Other extensions, e.g. for allowing agent communication, are not discussed in this paper.

In order to have an efficient intention selection function ($S_{\mathcal{I}}$ in Definition 8 and Figure 1, in Section 2.1.1), the idea that we are presently pursuing is the following. The DTC scheduler (briefly mentioned in Section 2.2) produces, for a given TÆMS task structure, alternative sequences in which an agent should execute the methods in that task structure so as to best satisfy the criteria (quality, duration, and cost) specified for them in the task structure of interrelated methods. Therefore, if we create a TÆMS task structure where the method labels are in fact the labels that uniquely identify the plans that are on the top of the intention stacks, and the programmer can set specific values for the scheduling criteria for each plan as well as the (TÆMS-like) relations of each plan to the others, then applying DTC to this task structure generates an order in which the plans that are candidate for intention selection would be best chosen for execution.

Further defining the translation of the set of intentions into TÆMS task structure, a TÆMS method label will in fact be a reference (through plan labels) to the intended

means[2] on the top of the stacks forming the agent's set of intentions. The labels of the intended means that are beneath the top of the stacks, will be used as intermediate task labels. The root task will have a conjunction of subtasks, one for each of the stacks of instantiated plans in the set of intentions. This provides a simple algorithm for translating an agent's current set of intentions into a TÆMS task structure.

One important alteration is needed in the data structures used to store the set of intentions. We now need to store the current scheduling criteria and relations for each intended means together with it in the set of intentions, so as to be able to generate the TÆMS task structure representing the particular state of the set of intentions. Note that the criteria and relations can change during the execution of a plan, as seen in the next section; that is why they have to be stored in the set of intentions.

Considering that we can translate the set of intentions into a TÆMS task structure, and we can run DTC on it, it is easy to create an efficient intention selection function for the AgentSpeak(XL) interpreter: it suffices to read the schedules of plan labels produced by DTC. All the intention selection function has to do is select the first formula (e.g., an action or a subgoal) in the body of the intended plan whose label is in the beginning of a schedule provided by DTC. When the formula being executed marks the end of a plan, that plan's label is also removed from the schedule.

4.3 The Task Structure Library

In this initial version of the Task Structure library of internal actions, which is accessed by the ts library name, we do not consider the possibility of a method having more than one *outcome*, and we define specific values rather than *distributions* for each of the DTC scheduling criteria (i.e., we consider the *density* to be the maximum one for each of these values). For the precise meaning of these TÆMS terms, please refer to [Wagner et al., 1998].

Each of the methods which represent intended means on top of intention stacks will have a single outcome (which can be associated to whether the plan finished successfully or not). Therefore, when the parameters for the scheduling criteria of that intended means are set (in the plan itself), they can only refer to that single outcome. In order to inform the AgentSpeak(XL) interpreter of the specific values for the three DTC scheduling criteria, we use the following internal actions (defined within the ts library):

$$ts.Criteria(quality\_value, duration\_value, cost\_value)$$

or the programmer can use these specific actions to set one particular criterion:

$$ts.Quality(value)$$

$$ts.Duration(value)$$

$$ts.Cost(value)$$

Note that the parameters to these internal actions can be variables if they have been previously instantiated in the plan.

[2]Recall that an intended means is a partially instantiated plan in the agent's set of intentions.

Also, the first version of the AgentSpeak(XL) ts library shall not make use of TÆMS quality accumulation functions other than q_sum.

The following internal actions are used for specifying the TÆMS *facilitates* and *hinders* relations between the plan that executes the action and those given as parameter (we are investigating whether the other relations should also be considered):

$$ts.Facilitates(P_1, \ldots, P_n)$$

$$ts.Hinders(P_1, \ldots, P_n)$$

It is interesting to note that, because these criteria and relations can be specified not only in the context but also in the body of plans, they can be changed dynamically by AgentSpeak(XL) programmers. The ts internal actions actually change the plan schedule criteria and plan relations that are associated with that intended means in the set of intentions.

There are two ways in which the TÆMS task structure generated from the agent's set of intentions can change: either when a plan executes actions from the ts library, which change the criteria and relations among plans, or when a new intended means is inserted in the set of intentions, which means that there is a new TÆMS "method" to be considered for scheduling). Both of them trigger the generation of a new TÆMS task structure for the current set of intentions and the execution of DTC on that task structure in order to get a new schedule of plan labels to be used by the intention selection function.

5 Conclusion

We have presented some ideas on how to integrate the DTC scheduler with an AgentSpeak(L)-like programming language, which we called AgentSpeak(XL). Some new constructs were added to the language, for its general improvement and to accommodate the DTC-based selection function. This allows the automatic conversion of the agent's set of intentions into a TÆMS task structure so that its DTC schedules define efficient intention selection functions. At this moment, we still do not have any quantitative evaluation of the improvement on the interpreter's performance resulting from this extension, but we expect a significant one. It is clearly more reasonable than, e.g., the simple one we used in [Machado and Bordini, 2001]. That quantitative evaluation should be reported in a paper soon.

The very next steps in this project are precisely related to such quantitative evaluation. There are also a number of questions to be answered, which will only be possible when we have the experimental data on which to make specific quantitative evaluations. We have to assess whether our decisions to schedule whole plans rather than basic (external) actions was a good one. We also have to investigate the use of DTC in other selection functions, in particular the one that selects among the options that there exist for applicable plans of a particular event. Improvements in the event selection function also have to be considered, either with DTC or by some other means.

Pursuing the specific ideas presented here, has been quite illuminating about the frameworks which we have been trying to integrate. In the longer term, we have to reconsider, preferably in a meeting between the Brazilian and US teams, our more ambitious

goals so as to set the agenda for working towards a full integration of GPGP and a BDI programming language in a two-level agent architecture.

References

[Decker, 1996] Decker, K. S. (1996). Tæms: A framework for analysis and design of coordination mechanisms. In O'Hare, G. M. and Jennings, N. R., editors, *Foundations of Distributed Artificial Intelligence*, chapter 16. John Wiley & Sons, Chichester.

[Decker, 1998] Decker, K. S. (1998). Task environment centered simulation. In Prietula, M., Carley, K., and Gasser, L., editors, *Simulating Organizations: Computational Models of Institutions and Groups*. AAAI Press / MIT Press, Menlo Park, CA.

[Decker and Lesser, 1995] Decker, K. S. and Lesser, V. R. (1995). Designing a family of coordination algorithms. In Lesser, V. and Gasser, L., editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95), 12–14 June, San Francisco, CA*, pages 73–80, Menlo Park, CA. AAAI Press / MIT Press. (Longer version available as UMass CS-TR 94-14.).

[Decker and Li, 1998] Decker, K. S. and Li, J. (1998). oordinated hospital patient scheduling. In Demazeau, Y., editor, *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98), Agents' World, 4–7 July, Paris*, Washington. IEEE Computer Society Press.

[d'Inverno et al., 1998] d'Inverno, M., Kinny, D., Luck, M., and Wooldridge, M. (1998). A formal specification of dMARS. In Singh, M. P., Rao, A. S., and Wooldridge, M., editors, *Intelligent Agents IV—Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97), Providence, RI, 24–26 July, 1997*, number 1365 in Lecture Notes in Artificial Intelligence, pages 155–176. Springer-Verlag, Berlin.

[d'Inverno and Luck, 1998] d'Inverno, M. and Luck, M. (1998). Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation*, 8(3):1–27.

[Durfee and Lesser, 1991] Durfee, E. H. and Lesser, V. (1991). Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183.

[Hindriks et al., 1999] Hindriks, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J.-J. C. (1999). Control structures of rule-based agent languages. In Müller, J. P., Singh, M. P., and Rao, A. S., editors, *Intelligent Agents V—Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), held as part of the Agents' World, Paris, 4–7 July, 1998*, number 1555 in Lecture Notes in Artificial Intelligence, pages 381–396, Heidelberg. Springer-Verlag.

[Kinny, 1993] Kinny, D. (1993). The distributed multi-agent reasoning system architecture and language specification. Technical report, Australian Artificial Intelligence Institute, Melbourne, Australia.

[Lesser, 1998] Lesser, V. R. (1998). Reflections on the nature of multi-agent coordination and its implications for an agent architecture. *Autonomous Agents and Multi-Agent Systems*, 1(1):89–111.

[Lesser et al., 1999] Lesser, V. R., Atighetchi, M., Benyo, B., Horling, B., Raja, A., Vincent, R., Wagner, T., Xuan, P., and Zhang, S. X. Q. (1999). A multi-agent system for intelligent environment control. In *Proceedings of the Autonomy Control Software Workshop (Autonomous Agent Workshop)*, Seattle.

[Lesser et al., 1998] Lesser, V. R., Decker, K., Carver, N., Garvey, A., Neiman, D., Nagendra Prasad, M., and Wagner, T. (1998). Evolution of the gpgp domain-independent coordination framework. Technical Report 98-05, University of Massachusetts, Amherst.

[Machado and Bordini, 2001] Machado, R. and Bordini, R. H. (2001). Running AgentSpeak(L) agents on SIM_AGENT. In *Pre-Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), August 1–3, 2001, Seattle, WA*. Proceedings to appear as a volume of LNCS – Intelligent Agents Series.

[Mayfield et al., 1996] Mayfield, J., Labrou, Y., and Finin, T. (1996). Evaluation of KQML as an agent communication language. In Wooldridge, M., Müller, J. P., and Tambe, M., editors, *Intelligent Agents II—Proceedings of the Second International Workshop on Agent Theories, Architectures, and Languages (ATAL'95), held as part of IJCAI'95, Montréal, Canada, August 1995*, number 1037 in Lecture Notes In Artificial Intelligence, pages 347–360, Berlin. Springer-Verlag.

[Móra et al., 1999] Móra, M. C., Lopes, J. G., Viccari, R. M., and Coelho, H. (1999). BDI models and systems: Reducing the gap. In Müller, J. P., Singh, M. P., and Rao, A. S., editors, *Intelligent Agents V—Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), held as part of the Agents' World, Paris, 4–7 July, 1998*, number 1555 in Lecture Notes in Artificial Intelligence, pages 11–27, Heidelberg. Springer-Verlag.

[Móra and Vicari, 2001] Móra, M. C. and Vicari, R. M. (2001). Integrating decision-making with formal BDI reasoning. Draft paper.

[Nagendra Prasad and Lesser, 1996] Nagendra Prasad, M. V. and Lesser, V. R. (1996). Learning situation-specific coordination in generalized partial global planning. In *AAAI Spring Symposium on Adaptation, Co-evolution and Learning in Multi-Agent Systems*, Stanford, CA. AAAI Press.

[Oates et al., 1997] Oates, T., Nagendra Prasad, M. V., and Lesser, V. R. (1997). Cooperative information gathering: A distributed problem-solving approach. *In IEEE Proceedings on Software Engineering, Special Issue on Agent-based Systems*, 144(1).

[Rao, 1996a] Rao, A. S. (1996a). AgentSpeak(L): BDI agents speak out in a logical computable language. In Van de Velde, W. and Perram, J., editors, *Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), 22–25 January, Eindhoven, The Netherlands*, number 1038 in Lecture Notes in Artificial Intelligence, pages 42–55, London. Springer-Verlag.

[Rao, 1996b] Rao, A. S. (1996b). Decision procedures for propositional linear-time belief-desire-intention logics. In Wooldridge, M., Müller, J. P., and Tambe, M., editors, *Intelligent Agents II—Proceedings of the Second International Workshop on Agent Theories, Architectures, and Languages (ATAL'95), held as part of IJCAI'95, Montréal, Canada, August 1995*, number 1037 in Lecture Notes In Artificial Intelligence, pages 33–48, Berlin. Springer-Verlag.

[Rao and Georgeff, 1995] Rao, A. S. and Georgeff, M. P. (1995). BDI agents: From theory to practice. In Lesser, V. and Gasser, L., editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95), 12–14 June, San Francisco, CA*, pages 312–319, Menlo Park, CA. AAAI Press / MIT Press.

[Rosenschein and Zlotkin, 1994] Rosenschein, J. and Zlotkin, G. (1994). *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, Cambridge, MA.

[Rossetti et al., 2001] Rossetti, R. J. F., Bordini, R. H., Bazzan, A. L., Bampi, S., Liu, R., and Van Vliet, D. (2001). Using BDI agents to improve driver modelling in a commuter scenario. *Journal of Transportation Research Part C: Special Issue on Emerging Technologies*. To appear.

[Singh et al., 1999] Singh, M. P., Rao, A. S., and Georgeff, M. P. (1999). Formal methods in DAI: Logic-based representation and reasoning. In Weiß, G., editor, *Multiagent Systems—A Modern Approach to Distributed Artificial Intelligence*, chapter 8, pages 331–376. MIT Press, Cambridge, MA.

[Sloman and Poli, 1996] Sloman, A. and Poli, R. (1996). SIM_AGENT: A toolkit for exploring agent designs. In Wooldridge, M., Müller, J. P., and Tambe, M., editors, *Intelligent Agents II—Proceedings of the Second International Workshop on Agent Theories, Architectures, and Languages (ATAL'95), held as part of IJCAI'95, Montréal, Canada, August 1995*, number 1037 in Lecture Notes In Artificial Intelligence, pages 392–407, Berlin. Springer-Verlag.

[Vincent et al., 2001] Vincent, R., Horling, B., Lesser, V., and Wagner, T. (2001). Implementing soft real-time agent control. In Müller, J. P., Andre, E., Sen, S., and Frasson, C., editors, *Proceedings of The Fifth International Conference on Autonomous Agents (Agents 2001), 28 May – 1 June, Montreal, Canada*, pages 355–362. ACM Press.

[von Martial, 1992] von Martial, F. (1992). *Coordinating Plans of Autonomous Agents*. Number 610 in Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin.

[Wagner et al., 1998] Wagner, T., Garvey, A., and Lesser, V. (1998). Criteria-directed heuristic task scheduling. *International Journal of Approximate Processing, Special Issue on Scheduling*, 19(1–2):91–118.

[Wellman and Doyle, 1992] Wellman, M. and Doyle, J. (1992). Modular utility representation for decision-theoretic planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 236–242.

[Wooldridge, 1999] Wooldridge, M. (1999). Intelligent agents. In Weiß, G., editor, *Multiagent Systems—A Modern Approach to Distributed Artificial Intelligence*, chapter 1, pages 27–77. MIT Press, Cambridge, MA.

[Wooldridge, 2000] Wooldridge, M. (2000). Computationally grounded theories of agency. In Durfee, E., editor, *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000), 10–12 July, Boston*, pages 13–20, Los Alamitos, CA. IEEE Computer Society. Paper for an Invited Talk.