The ARTS Real-Time Agent Architecture

Konstantin Vikhorev
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
Email: kxv@cs.nott.ac.uk

Natasha Alechina
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
Email: nza@cs.nott.ac.uk

Brian Logan
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
Email: bsl@cs.nott.ac.uk

***Abstract*—We present a new approach to providing soft real-time guarantees for Belief-Desire-Intention (BDI) agents. We define what it means for BDI agents to operate in real time, or to satisfy real-time guarantees. We then develop a model of real time performance which takes into account the time by which a task should be performed and the relative priority of tasks, and identify the key stages in a BDI architecture which must be bounded for real time performance. As an illustration of our approach we introduce a new BDI architecture, ARTS, which allows the development of agents that guarantee (soft) real time performance. ARTS extends ideas from PRS and JAM to include goals and plans which have deadlines and priorities, and schedules intentions so as to achieve the largest number of high priority intentions by their deadlines.**

***Index Terms*—BDI Agent, Real-time guarantees, Task Scheduling, Priority, Deadline, Agent Real-Time System, ARTS.**

I. Introduction

The design of an agent system which can operate effectively in a real-time dynamic environment is a major challenge for multiagent research. The main difficulty in building real-time agent systems is how to specify real-time constraints and how to ensure that the agent system meets these constraints. As with other computational systems, agents are *resource bounded* because their processors have limited speed and memory. Traditionally, agents have been developed without much attention to resource limitations. However such limitations become important when an agent system operates in a dynamic environment. The reasoning processes implicit in many agent architectures may require significant time to execute (in some cases exponential time), with the result that the environment may change while the agent makes a decision about which activity to pursue. Thus a decision made by the agent may be wrong (incorrect, sub-optimal, or simply irrelevant) if it is not made in a timely manner.

A number of agent architectures and platforms have been proposed for the development of agent systems which must operate in highly dynamic environments. For example, the Procedural Reasoning System (PRS) [1] and PRS-like systems, e.g., PRS-CL [2], JAM [3], SPARK [4] have features such as metalevel reasoning which facilitate the development of agents for real time environments. However, to provide real time guarantees, these systems have to be programmed for each particular task environment—there are no general methods or tools which allow the agent developer to specify that a particular goal should be achieved by a specified time or that an action should be performed within a particular interval of an event occurring. Rather each application has to be individually tuned by the developer. There are also a number of hybrid agent architectures such as ROACS [5] and SIMBA [6]. A hybrid architecture consists of an AI subsystem and a low-level control subsystem connected by communication interface. Such systems attempt to improve responsiveness by separating the 'real-time' aspects of the architecture from the high-level control. However while such systems can simplify the development of agents for real-time environments, they provide limited high-level support for managing the timely execution of tasks.

In this paper we present a new approach to Belief-Desire-Intention (BDI) architectures for real-time agents. We develop a model of real time performance which takes into account the time by which a task should be performed and the relative priority of tasks, and identify the key stages in a BDI architecture which must be bounded for real time performance. As an illustration of our approach we introduce a new BDI architecture, ARTS, which allows the development of agents that guarantee (soft) real time performance. ARTS extends ideas from PRS and JAM to include goals and plans which have *deadlines* and *priorities*, and schedules intentions so as to achieve the largest number of high priority intentions by their deadlines.

The remainder of the paper is organised as follows. In section 2 we develop a notion of 'real-time' appropriate to agent-based systems. In section 3 we present our method for any BDI architecture. In section 4 to illustrate our approach the new real-time agent architecture ARTS is introduced. In section 5 we compare our methods with related approaches. And, finally, in section 6 we conclude and outlined directions for future research. discussion.

II. Real-Time Guarantees

In real-time programming a distinction is made between *hard real-time* and *soft real-time* systems. In the context of agent systems, hard real-time means that the agent must process its inputs (i.e., facts and goals) and produce a response within a specified time. For an agent system which provides hard real-time guarantees there is therefore a strict upper bound on the time to process incoming information and produce a response. In soft real-time, the agent may not produce a response within the specified time in all cases, i.e. timeliness constraints may be violated under load and fault conditions without critical consequences.[1] For BDI agents, we would argue that the relevant notion of 'response' is the achievement of a high level goal. However, for agents in open environments, providing hard real-time guarantees for anything other than the internal operations of the agent is typically not feasible, unless we make strong assumptions about the characteristics of the agent's environment. In this paper we therefore focus on soft real-time guarantees for achieving the agent's top level goals.

We assume that each of the agent's top level achievement goals is associated with a (possibly infinite) *deadline* which specifies the time by which the goal should be achieved. A set of goals which can all be achieved by the deadlines is termed *feasible*. Which sets of goals are feasible will depend on the speed at which the environment changes, the capabilities of the agent etc. In general, it may not be possible to achieve all of an agent's goals by their deadlines. For example, goals produced by users or other agents, or autonomously generated in response to an event in the agent's environment, may result in a previously feasible set of goals becoming infeasible, if there is insufficient time to achieve each goal, or an agent may have no plan to achieve a particular goal. In such situations, it is frequently more important to achieve some goals than others. For example, the goal of submitting a conference paper on time may be more important than a goal to get coffee. We therefore assume that each goal is associated with a *priority* which specifies the importance of achieving the goal. Priorities define a total preorder, $\preceq$, over goals. A set of goals g is said to be maximal if it is feasible and there is no other set of goals g' such that $g' \preceq g$ for some suitable lifting of $\preceq$ to sets of goals. We define a *real-time BDI agent* as an agent which achieves a maximal set of goals, i.e., the largest number of high priority goals by their deadlines.

III. Changes to the BDI Architecture

In this section we outline the changes necessary to a BDI architecture to implement a *real-time BDI agent*. We assume a simple generic BDI architecture in which an agent has beliefs and goals, and selects plans (sequences of subgoals and primitive actions) in order to achieve its goals or in response to new beliefs. Once the agent has adopted a plan it becomes an intention, and at each cycle the agent executes a single step of one of its current intentions. To implement real-time BDI agents within such an architecture, two main changes are required: we must add additional information about goals and plans required to support real time guarantees, and we need to change the BDI execution cycle to ensure that the agent's cycle time is bounded and that the maximum number of high priority goals are achieved by their deadlines. We consider each in turn below.

[1]Some computer systems (for example, real-time video) utilise a stricter notion of real-time guarantees, where the precise time at which a response is produced matters [7], [8]. Hard real-time for this type of system requires a response at an exact time rather than before a deadline, and soft real-time means that the response time lies within a defined uncertainty range around the required time. However we would argue that, in agent based systems, this stricter sense of real time guarantee is less appropriate; for many task environments, the key requirement is that a goal is achieved *before* a deadline rather than exactly at a specific time.

A. Additional Information

As discussed above, in order to provide real-time guarantees, each top-level goal must be associated with a deadline which specifies the time by which the goal should be achieved. We assume that the deadline for a goal is specified when the goal is generated by a user (or another agent), and is expressed as a real time value in some appropriate units (milliseconds, minutes, hours etc.).[2] By default, the plan selected to achieve a top-level goal (and its subgoals and subplans) inherit the deadline of the top-level goal. However we allow the deadline of the top-level goal to be advanced by a plan, if the execution context of the plan is such as to suggest that an earlier deadline should be adopted for the goal. For *fact-invoked* plans (i.e., plans triggered by the agent's beliefs), the deadline is infinity.

Each top-level goal is also associated with a priority (e.g., a non-negative integer value, with larger values taken to indicate higher priority) which specifies the relative importance of achieving the goal. Each plan also has a *plan priority* which specifies the relative utility of the plan for the triggering goal or belief. We assume that the agent always intends plans with the highest priority and that goal and plan priorities are commensurable.

Each plan is also associated with a *duration*, an estimate of the real time necessary to execute the plan. In order to define durations, we assume that each primitive action has a *timeout* which specifies the maximum amount of real time required to perform the action. Actions which do not complete by their timeout are assumed to have failed. The duration of a non-primitive activity within a plan is the sum of the durations of its subplans (i.e., the duration of a top-level plan is the sum of the durations of all subplans intended for the goal). Assuming the plan library is of fixed size, we compute the durations of subplans as follows:

1) For every agent's plan, we compute all possible variants of an intention, leading by this plan. This can be represented as a tree structure. For the moment, we assume that there are no loops or recursion within plans.
2) Leaf plans do not contain calls to other plans and include only the addition and deletion of goals and primitive actions, and their duration can be easily calculated from the time required for basic agent actions (see below) and the timeouts on primitive actions.
3) Starting from leaf plans we can estimate the duration of each intention variant. The maximum and the minimum duration are the upper and the lower bound of the plan duration.

In case of plans with loops with undefined number of repetitions or recursion within the plan, the minimum duration is the shortest path through the tree structure and the maximum duration is infinity. In most cases, especially in a complex

[2]For simplicity, we do not consider agents which can generate their own top-level goals—all top-level goals are therefore external.

system, we will not able to provide the exact upper bound estimation of duration.

B. Changes to the BDI Execution Cycle

We assume that the internal operations of the agent—adding or deleting a belief or goal, selecting a plan, adopting an intention, selecting an intention to execute and executing a single step of the intention—require time bounded by the size of the agent's program and its beliefs and goals. Adding or deleting a belief or goal, adopting an intention, and executing a single step of an intention can be assumed to take constant time. However selecting a plan and intention to execute are intractable in the general case, and it is necessary to approximate the choices of an unbounded agent to limit the agent's cycle time.

To bound the time necessary to select a plan, we assume that goals and plans are processed in order of priority. That is, for each goal in priority order, the highest priority plan for that goal is checked to see if it is both executable in the current belief context and feasible (has a duration less than the deadline of the triggering goal). If the plan is executable and feasible, the agent immediately commits to the plan and processing moves to the next goal. If the plan is not executable or feasible matching continues for the current goal with the next plan in priority order. Plan selection stops when a user definable *plan selection timeout* is reached. At this point the agent has zero or more executable, feasible plans, which are merged into the intention structure, either as new top-level intentions (for plans triggered by new top-level goals or facts), or are added to existing intentions.

To bound the time necessary to select an intention to execute at the current cycle, we utilise a deadline monotonic scheduling algorithm which, while not optimal, gives preference to urgent, high-priority intentions:

1) find the highest priority feasible intention, i.e., where the remaining execution time is less than the deadline;
2) find the next most important intention which is feasible for the existing schedule and assuming that tasks are executed in deadline order, earliest deadline first;
3) repeat 2 until no more intentions can be scheduled;
4) execute the next step of the first intention in the schedule.

An intention is feasible if it can be inserted in the schedule in deadline order while meeting its own and all currently scheduled deadlines. If all intentions in the schedule had the same priority, then the resulting schedule must be feasible if any schedule is, i.e., if a system is unschedulable with deadline monotonic ordering then it is unschedulable with all other orderings [13]. This algorithm has a worst case complexity of O(n), where n is the number of the agent's intentions.

There are two possible scheduling scenarios: pessimistic, which is based on the upper bound of duration and optimistic, which is based on the lower bound of duration. Pessimistic scheduling has limited applicability, as in most cases the upper bound on duration is equal to infinity. In many cases, it is therefore more reasonable to implement an optimistic scheduler as this places no restrictions on the structure of plans.

IV. ARTS: Agent Real-Time System

In this section ARTS, an implementation of the real-time BDI agent architecture described above. ARTS is an agent programming framework for agents with soft real-time guarantees; an ARTS agent will attempt to achieve as many high priority tasks by their specified deadlines as possible. The syntax and execution semantics of ARTS is based that of PRS-CL [2] and JAM [3], augmented with information about deadlines, priorities, and durations, and changes to the interpreter to implement time bounded priority driven plan selection and deadline monotonic intention scheduling. ARTS is implemented in Java, and the current prototype implementation includes the core language described below, and implementations of some basic primitive actions. Additional user-defined primitive actions can be added using a Java API. In the interests of brevity, we have omitted the meta-level features of ARTS.

An ARTS agent consists of five main components: a database, a goal stack, a plan library, an intention structure, and an interpreter. The database contains the agent's current beliefs (facts). The goal stack is a set of goals to be realised. The plan library contains a set of plans which can be used to achieve agent's goals or react to particular situations. The intention structure contains plans that have been chosen to achieve goals or respond to facts. The interpreter is the main component of the agent. It manipulates the agent's database, goal stack, plan library and intention structure and reasons about which plan to select based on the agent's beliefs and goals to create and execute intentions. Changes to the agent's environment or posting of new goals invokes reasoning to search for plans that might be applied to the current situation. The ARTS interpreter selects one plan from the list of applicable plans, intends and schedules it, and executes the next step of first intention in the computed schedule.

A. Facts

The database of an ARTS agent contains facts (beliefs) that represent the state of the agent and its environment. Facts may represent information about state variables, sensory input, derived information or information about other agents, etc.

```
fact       ::=  wff
wff        ::=  pred_name term_exp* | (NOT wff) | (AND wff+)
                | (OR wff+)
term_exp   ::=  value | variable | function
value      ::=  integer | float | string
variable   ::=  "$"var_name
function   ::=  (fun_name term_exp+)
```

where *pred_name*, *fun_name* and *var_name* name predicated, functions and variables respectively.

B. Goals

ARTS distinguishes two categories of goals: top-level goals and subgoals. ARTS supports two top-level goal operators:

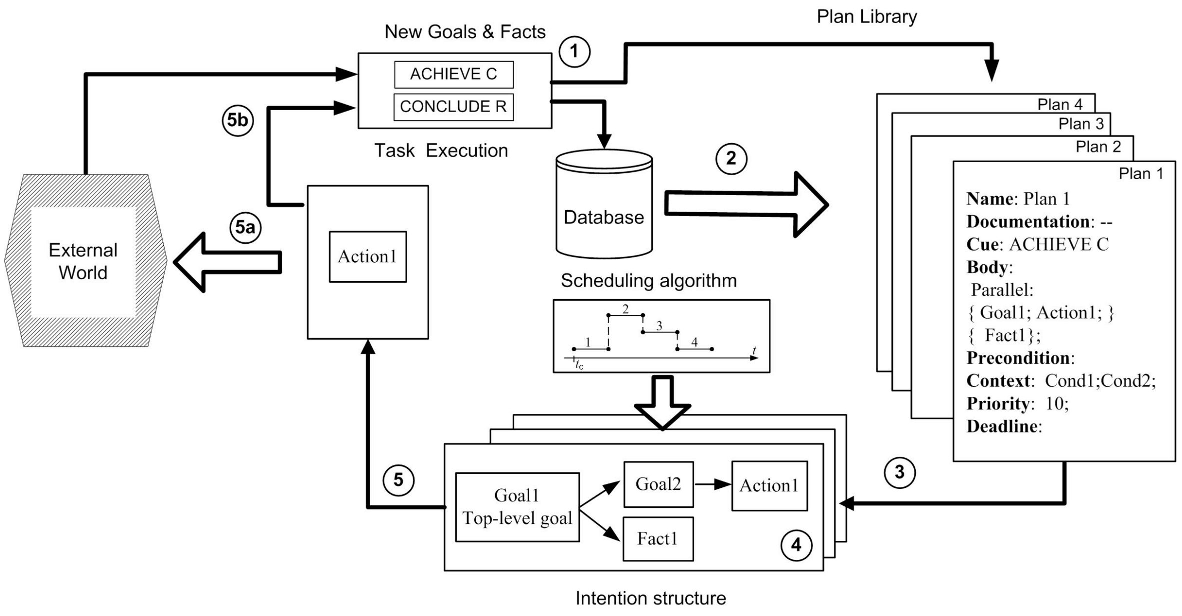

Fig. 1. The execution cycle of ARTS agent

ACHIEVE and CONCLUDE. An ACHIEVE goal specifies that the agent desires to achieve a particular goal state. A CONCLUDE goal inserts a certain fact into the database and possibly invokes fact-driven plan. The form of top-level goals is given by:

goal_exp	::=	*achieve* | *conclude*
achieve	::=	"ACHIEVE" *wff* [*pr*] [*dl*] {[*by*] | [*not_by*]}";"
conclude	::=	"CONCLUDE" *wff* {[*by*] | [*not_by*]} ";"
pr	::=	":PRIORITY" *ground_term*
dl	::=	":DEADLINE" *ground_term*
by	::=	":BY" *plan_name*$^+$
not_by	::=	":NOT_BY" *plan_name*$^+$

where *plan_name* is the name of a plan. The :PRIORITY field is optional and allows the specification of either a constant priority or an expression which allows the calculation of the plan priority as function of plan variables (see below). The default priority of a top-level goal is zero. The :DEADLINE expression is also optional and allows the specification of the deadline of the goal. By default the deadline is equal to infinity. CONCLUDE goals have zero priority and an infinite deadline.

The developer can specify one or more top-level goals for the agent as part of the agent's program using the keyword "GOALS:". For example:

```
GOALS:
  ACHIEVE PrepareLecture agents101 : PRIORITY 9 :DEADLINE 50;
  ACHIEVE HaveLunch :PRIORITY 7 :DEADLINE 40;
  ACHIEVE BorrowBook R&N :PRIORITY 2 :DEADLINE 30;
```

Subgoals are goals generated within plans. ARTS has the following subgoals operators:

ACHIEVE C	achieve condition C
CONCLUDE F	add fact F to the database
TEST C	test for the condition C
RETRACT F	retract fact F from database
WAIT C	wait until condition C is true

In contrast to top-level goals, the deadline and priority of ACHIEVE subgoals are not specified but inherited from the plan.

C. Plans

Plans define a procedural specification for achieving a goal. In specifying plans we distinguish between *plan trigger variables* and *plan body variables*. Plan trigger variables are free variables appearing in the cue, precondition and context fields, while plan body variables are variables appearing in the body of the plan. Plan trigger variables must be ground when the plan is selected, while binding of plan body variables can be deferred to the execution of the corresponding plan step. The agent's plan library is introduced by the keyword "PLANS:" followed by a list of plans of the form:

Name is an unique symbolic identifier of the Plan.

Documentation is an optional field which used to store a descriptive text string.

Cue specifies the purpose of the Plan and is used to select the plan for possible execution. The **Cue** field can contain either an ACHIEVE or CONCLUDE goal. A ACHIEVE goal in the **Cue** field means that the Plan may be used to achieve some condition, while a CONCLUDE goal means that the Plan may be chosen for possible execution when a fact is added to the database.

Precondition specifies conditions that must be satisfied for Plan to be applicable. This field is optional and can contain both ACHIEVE and TEST goal expressions. An ACHIEVE **G** precondition means that the system must currently have **G** as a goal in order for the Plan to be applicable, while a TEST **C** precondition means that **C** must be true for the Plan to be applicable.

Context defines additional conditions (i.e. ACHIEVE and TEST goal expressions) on plan execution. This field is optional and has similar functionality to the Precondition field, but in contrast to the precondition it must be satisfied before and during Plan execution. As in JAM, this significantly increases the reactivity of the agent.

Body defines a sequence of simple activities (i.e. primitive actions, addition and deletion of goals and facts), and complex constructs (e.g. loops, (non)deterministic choice, etc, see below).

Priority specifies the relative utility of the Plan. The plan priority is used to choose between the applicable plans for a particular goal. The priority field is optional and allows the specification of either a constant priority or an expression which allows the calculation of the plan priority as function of variables appearing in the plan trigger. The default priority value is 0.

Deadline specifies a deadline for the plan. The deadline field is optional and allows programmer to advance the deadline inherited from the triggering goal. The deadline can be specified as a constant value or an expression which allows the calculation of the plan deadline as function of variables appearing in the plan trigger. If the specified plan deadline is earlier than the deadline for this intention it becomes the deadline for the intention during the execution of the plan (i.e., it effectively advances the deadline for this intention during the execution of the plan). If the specified deadline is later than the deadline for the intention, the plan deadline is ignored.

ARTS, like JAM, supports standard programming constructs such as DO ...WHILE (loop with postcondition) and WHILE construct (loop with precondition), choice constructs specified by OR (do any in order), AND (do all in order), DO_ALL (do all randomly), DO_ANY (do any randomly), WHEN (conditional execution), and ASSIGN (assignment to plan body variables). The BNF for plans is given in table I.

D. Primitive Actions

The subgoal operators are implemented directly by the ARTS interpreter. Other primitive actions are implemented as Java methods. Each primitive action referenced in a plan body must have Java code which implements the necessary functionality. ARTS supports two mechanisms for defining primitive actions: writing a class which implements the PrimitiveAction interface, and direct invocation of methods in existing legacy Java code. Primitive actions are executed by using an EXECUTE action.

In contrast to PRS-CL and JAM, ARTS allows the agent programmer to specify a timeout for each primitive action by using the TIMEOUT keyword. The timeout specifies the maximum amount of real time required to perform the action. Actions which do not complete by their timeout are assumed

plan	::=	"PLAN: {"*p_name* [*p_doc*] *p_cue* [*p_precond*] [*p_cont*] *p_body* [*p_pr*] [*p_dl*] [*p_attr*] "}"
p_name	::=	"NAME:" string";"
p_doc	::=	"DOCUMENTATION:" [string]";"
p_cue	::=	"CUE:" *p_goal_exp* ";"
p_precond	::=	"PRECONDITION:" *p_cond** ";"
p_cont	::=	"CONTEXT:" *p_cond** ";"
p_body	::=	"BODY:" *body_elem**
p_pr	::=	"PRIORITY":" *term_exp* ";"
p_dl	::=	"DEADLINE":" *term_exp* ";"
body_seq	::=	"{" *body_elem** "}"
body_elem	::=	*activity* | *b_and* | *b_or* | *b_parallel* | *b_do_all* | *b_do_any* | *b_do_while* | *b_while* | *b_when*
activity	::=	*prim_act* | *misc_act* | *subgoal* ";"
b_and	::=	"AND:" *body_seq*$^+$";"
b_or	::=	"OR:" *body_seq*$^+$";"
b_parallel	::=	"PARALLEL:" *body_seq*$^+$";"
b_do_all	::=	"DO_ALL:" *body_seq* $^+$";"
b_do_any	::=	"DO_ANY:" *body_seq*$^+$";"
b_do_while	::=	"DO:" *body_seq* "WHILE:" *p_cond*";"
b_while	::=	"WHILE:" *p_cond body_seq* ";"
b_when	::=	"WHEN:" *p_cond body_seq*";"
p_goal_exp	::=	"ACHIEVE" *wff* | "CONCLUDE" *wff*
p_cond	::=	"ACHIEVE" *wff* | "TEST" *wff*
subgoal	::=	*subgoal_op wff*";"
subgoal_op	::=	"ACHIEVE" | "CONCLUDE" | "TEST" | "RETRACT" | "WAIT"
prim_act	::=	"EXECUTE:" *function* [":TIMEOUT" *ground_term*]
misc_act	::=	"ASSIGN:" *term_exp term_exp*

TABLE I
PLAN BNF

to have failed. For example:

EXECUTE *move_to* **$x** **$y** :TIMEOUT 50

E. Interpreter

The ARTS interpreter repeatedly executes the set of activities shown in Figure 1.

1) New goals are added to the goal stack and facts corresponding to CONCLUDE goals and external events are added to the database.
2) The precondition and context expressions of plans with a cue matching a goal on the goal stack are evaluated against the database to determine if the plan is applicable in the current situation. Goals and plans are matched in priority order as described in section III-B. For ACHIEVE goals, the interpreter checks to see whether the goal has already been accomplished before trying to invoke a plan.
3) The resulting set of applicable plans are placed on the intention intention structure.
4) Intentions are scheduled according to their deadline and priority value as described in section III-B. Intentions which are not schedulable, i.e., their minimum remaining execution time is greater than the time remaining to their deadline, are either dropped or have their priority reduced to zero.[3]
5) Finally, the interpreter selects the first intention from the computed schedule and executes the one step of

[3]This choice is currently determined by a global flag, rather than per goal.

that intention. The result of the execution can be (5a) execution of a primitive action or (5b) the posting of a new subgoal or the conclusion of some new fact.

For ACHIEVE goals, the interpreter checks to see whether the goal has already been accomplished before trying to invoke a plan.

F. Example

In this section sketch as simple example ARTS agent and show how it allows the specification of soft real-time requirements. The agent has three goals: preparing a lecture, having lunch and picking up a book from the library. Each task has a different priority and deadline. For simplicity, we assume that actions never fail and that the unit of time is the minute.

```
GOALS:
  ACHIEVE PrepareLecture agents101 :PRIORITY 9 :DEADLINE 50;
  ACHIEVE HaveLunch :PRIORITY 7 :DEADLINE 40;
  ACHIEVE BorrowBook R&N :PRIORITY 2 :DEADLINE 30;

  CONCLUDE LectureNotes agents101 myNotes;

PLAN: {
  NAME: "Plan 1";
  DOCUMENTATION:  "Prepare for lecture";
  CUE:  ACHIEVE PrepareLecture $x, y;
  PRECONDITION:   TEST LectureNotes $x, y;
  BODY:
    EXECUTE revise-lecture $y :TIMEOUT 35;
}

PLAN: {
  NAME: "Plan 2";
  DOCUMENTATION:  "Pickup a book from the library";
  CUE:  ACHIEVE BorrowBook $x;
  BODY:
    EXECUTE goto library :TIMEOUT 10;
    ACHIEVE Pickup $x;
}

PLAN: {
  NAME: "Plan 3";
  DOCUMENTATION:  "Pick up something";
  CUE:  ACHIEVE Pickup $x;
  BODY:
    EXECUTE pickup $x :TIMEOUT 2;
}

PLAN: {
  NAME: "Plan 4";
  DOCUMENTATION:  "Have lunch";
  CUE:  ACHIEVE HaveLunch;
  BODY:
    EXECUTE eat-sandwich :TIMEOUT 20;
}
```

EXAMPLE ARTS AGENT

The algorithm for estimating the duration of a plan is executed when the agent is initialised. Plans have following maximum and minimum durations: $d_1 = 35min$, $d_2 = 12min$, $d_3 = 2min$, $d_4 = 20min$. Once the durations of plans have been estimated, the agent begins the reasoning cycle. The interpreter parses the initial top-level goals. It then attempts to match them against the plan library in order to invoke suitable plans. As a result **Plan 1**, **Plan 2** and **Plan 4** are added to the intention structure. As explained above, plans inherit their deadline and priority values from the triggering goal. This means that the intention related to the prepare lecture goal has the highest priority (9), the intention which corresponds to the goal to have lunch has medium priority (7), and the last intention related to the goal to pickup a book from the library has the lowest priority (2). The scheduling algorithm checks feasibility of each intention before adding them to the schedule. The first most important intention is inserted into schedule, because it is currently empty. Then the feasibility of the *HaveLunch* intention is checked. It is obvious that the intention is infeasible, because it can't be inserted to the schedule in deadline order together with first intention and it will be dropped. On the other hand the low priority intention is feasible and can be scheduled in deadline order together with the first one. It is important to note that the low priority task will be executed first, because it has an earlier deadline. Once the schedule has been computed, the interpreter executes one step of the first task, i.e., *goto* primitive action, and starts a new cycle.

At the second cycle the interpreter executes next step, i.e. the ACHIEVE *Pickup* goal. The goal invokes **Plan 3** which inherits the deadline of 30 and priority of 2 from the top-level goal and extends the existing intention. The interpreter then performs one step of the plan 3. In the same way it performs action to revise the notes for the lecture from the next intention.

V. RELATED WORK

The scheduling of intentions is key to realisation of real-time agents and a variety of intention scheduling approaches have been explored in the literature. For example, AgentSpeak(XL) [10] and the Soft Real-Time Agent Architecture [9] use the TÆMS (Task Analysis, Environment Modelling, and Simulation) domain-independent framework [11] together with Design-To-Criteria scheduling[12]. The TÆMS framework assumes a worth-oriented environment, in which goals are associated with a degree of achievement (i.e., a goal may be not fully achievable or not achievable at all). The TÆMS framework allows modelling tasks with deadlines. The problem of using DTC for scheduling agent intentions is that an agent which implements DTC will not perform tasks in some fixed order and is unable to compute a set of feasible tasks because the decision about which task (intention) will be executed is based on some rating (real value between 0 and 1), which changes from cycle to cycle.

There has been considerable work on approaches to the development of reasoning systems which must operate in highly dynamic environments, e.g., [15], [17], [2], [3], [4]. Much of this work has focused on deliberation and reasoning strategies involving metalevel plans and facts for reasoning about applicable plans, the failure to achieve a goal, changing the intention structure according to user specified criteria, etc. While metalevel reasoning provides great flexibility to the agent developer, it can be complex and has to be programmed for each particular application. In contrast, ARTS has its own well defined real-time reasoning mechanism for tasks with different priorities and deadlines, which does not require utilisation of metalevel capabilities.

VI. Conclusion

The main contributions of this paper are an analysis of the meaning of real-time guarantees for a BDI agent, and a proposal for a new BDI agent architecture, ARTS, for the development of real-time BDI agents. ARTS is influenced by the PRS family architectures, such as PRS-CL and JAM. However, unlike previous PRS-like architectures, ARTS includes a duration estimation algorithm, priority driven plan selection and a deadline monotonic intention scheduling algorithm. These features enable an ARTS agent to produce an intention schedule which achieves the greatest number of high priority goals by their deadlines. While the resulting schedule is not necessarily optimal, it is computable in bounded time, and we believe that the kind of "optimistic bounded rationality" implemented by the ARTS architecture provides a simple, predictable framework for agent developers, facilitating the development of agents which can perform tasks of different complexity and scale while providing timely responses to events in highly dynamic environments.

The current ARTS implementation has a number of limitations. For example, the architecture currently assumes that the agent must wait for the completion of each plan step before recomputing the intention structure, i.e., the agent can't execute intentions in parallel. For plans containing asynchronous primitive actions or `WAIT` goals, this is clearly not the case. In future work, we plan to extend the scheduler to handle asynchronous execution of intentions. Other directions for future work include improved algorithms for duration estimation and improvements to the basic BDI interpreter to reduce the overall cycle time.

References

[1] M. P. Georgeff and A. L. Lansky, "Procedural knowledge," in *IEEE*, vol. 74, no. 10. IEEE Press, 1987, pp. 1383–1398.

[2] K. L. Myers, *PRS-CL: A Procedural Reasoning System. User's Guide.*, SRI International, Center, Menlo Park, CA, March 2001.

[3] M. J. Huber, "JAM: A BDI-theoretic mobile agent architecture," in *Proceedings of The Third International Conference on Autonomous Agents*, Seattle, WA, 1999, pp. 236–243.

[4] D. Morley and K. Myers, "The spark agent framework," in *Proc. of the Third Int. Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS-04)*, New York, NY, July 2004, pp. 712–719.

[5] J. S. Gu and C. W. de Silva, "Development and implementation of a real-time open-architecture control system for industrial robot systems," *Engineering Applications of Artificial Intelligence*, vol. 17, no. 5, pp. 469 – 483, 2004.

[6] C. Carrascosa, J. Bajo, V. Julian, J. M. Corchado, and V. Botti, "Hybrid multi-agent architecture as a real-time problem-solving model," *Expert Systems Applications*, vol. 34, no. 1, pp. 2–17, 2008.

[7] J. Chakareski, J. Apostolopoulos, and B. Girod, "Low-complexity rate-distortion optimized video streaming," in *Proceedings of the International Conference on Image Processing (ICIP)*, vol. 3, Oct. 2004, pp. 2055–2058.

[8] S. G. Deshpande, "High quality video streaming using content-awareadaptive frame scheduling with explicit deadlineadjustment," in *MM '08: Proceeding of the 16th ACM international conference on Multimedia*. New York, NY, USA: ACM, 2008, pp. 777–780.

[9] R. Vincent, B. Horling, V. Lesser, and T. Wagner, "Implementing soft real-time agent control," in *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*. New York, NY, USA: ACM, 2001, pp. 355–362.

[10] R. Bordini, A. Bazzan, R. de, O. Jannone, D. Basso, R. Vicari, and V. Lesser, "Agentspeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling," in *In Proc. of AAMAS'02*, 2002, pp. 1294–1302.

[11] K. S. Decker and V. R. Lesser, "Quantitative modeling of complex environments," *International Journal of Intelligent Systems in Accounting, Finance and Management*, vol. 2, p. 215234, 1993.

[12] T. Wagner, A. Garvey, and V. Lesser, "Criteria-directed heuristic task scheduling," *International Journal of Approximate Reasoning*, vol. 19, pp. 91–118, Jyly 1998.

[13] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *JACM*, vol. 20, no. 1, pp. 46–61, 1973.

[14] R. T. Dodhiawala, N. S. Sridharan, P. Raulefs, and C. Pickering, "Real-time ai systems: A definition and an architecture," in *IJCAI*, 1989, pp. 256–264.

[15] M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning," in *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87*, 1987, pp. 677–682.

[16] M. P. Georgeff and F. F. Ingrand, "Real-time reasoning: Monitoring and control of spacecraft systems using procedural reasoning," in *Australian Artificial Intelligence Institute*, vol. 1, no. 03, Melbourne, Australia, May 1989.

[17] ——, "Managing Deliberation and Reasoning in Real-Time Systems," in *In Proceedings of the DARPA Workshop on Innovative Approaches to Planning*, San Diego, California, 1990.

[18] D. J. Musliner, J. Hendler, A. K. Agrawala, E. H. Durfee, J. K. Strosnider, and C. J. Paul, "The challenges of real-time AI," University of Maryland, Tech. Rep. CS-TR-3290, 1994.