Plexil-Like Plan Execution Control in Agent Programming

Pouyan Ziafati
SnT, University of Luxembourg
Intelligent Systems Group, Utrecht University

Abstract

BDI-based agent programming languages are well-known technologies for implementing autonomous agents in dynamic environments. Supporting robot programming however requires the plan representation and execution control capabilities of these languages to be extended for 1-) controlling and monitoring the execution of actions in complex arrangements and 2-) coordinating the parallel execution of plans over shared resources. To this end, this paper adapts and extends PLEXIL, an expressive and well-defined robotic plan execution language, for plan representation and execution in BDI-based agent programming languages. The syntax and semantics of the new language is presented and its integration in operational semantics of BDI-based agent programming languages is discussed.

Introduction

In order to achieve complex goals in dynamic environments, a robot needs to reason on its objectives and the state of its environment to select appropriate course of actions. Various agent programming languages (APLs) (Bordini et al. 2006) have been developed to facilitate the implementation of such deliberative behaviour based on the well studied BDI (Belief-Desire-Intention) model of practical reasoning(Bratman 1999). An agent operation in BDI architecture (Rao and Georgeff 1995; 1991) is a cyclic execution of a so called deliberation cycle in which the agent processes its input data, updates its goals and beliefs, applies a set of plan generating rules to plan upon its goals and beliefs and executes some of its planned actions. To achieve its goals, BDI agent usually does not plan from the scratch, but selects from a set of plan templates and instantiate them based on its context (i.e. goals and beliefs). Such "reactive planning" capability makes BDI-based agent programming languages particularly useful for programming agents such as robots operating in dynamic environments.

While agent programming languages provide a suitable level of abstraction and programming support for implementing deliberative behaviour, the plan representation and execution capabilities of these languages needs to be extended to facilitate their applications in robotics. Robot programming requires these languages to support (Ziafati et al. 2013) 1-) hierarchical task decomposition and controlling and monitoring the plan execution at different levels of plan hierarchy, 2-) supporting conditional contingencies, loops, temporal constraints and floating contingencies (i.e. event driven task execution) in the task tree decomposition, 3-) coordinating the parallel execution of plans over shared resources by their priorities and deadlines and 4-) performing clean-up and wind-down activities when pre-empting, aborting, suspending or resuming plans.

In order to extend the plan execution capabilities of agent programming languages, we opt to build upon the *PLEXIL*(Verma et al. 2005; Gilles et al. 2008) plan execution language developed in *NASA* due to the following reasons. *PLEXIL* offers a simple structure for plan representation, a hierarchy of nodes with few syntactic constructs, but it is one of the most expressive plan execution languages unifying many of the existing ones. Moreover, *PLEXIL* has formal semantics which allows for the formal study of various types of determinism of plan execution. In addition, the operational semantics of *PLEXIL* is presented in a modular way at various levels of plan execution easing the formal study and modification of the language. Finally, the languages has been successfully used in various robotic applications.

This paper adapts the *PLEXIL* syntax and semantics to be integrated in BDI-based agent programming languages for representing and executing plans. This includes introducing basic execution nodes for querying and manipulating the agent's beliefs and goals in the BDI architecture and presenting an operational semantics for *PLEXIL*-like plan execution in BDI deliberation cycle. Moreover, *PLEXIL* is extended to support pausing, resuming and pre-empting plans, performing clean-up and wind-down activities when pausing, resuming, pre-empting or aborting plans, and coordinating the parallel execution of plans over shared resources.

The remainder of this paper is as follows. We first provide a short introduction of BDI-based agent programming languages, focusing on main aspects of the BDI architecture rather than language specific features of such languages, and present a proposal for *PLEXIL*-like plan execution in BDI architecture. Then we present syntax and semantics of a *PLEXIL*-like language for plan representation and execution in agent programming languages. Finally we present future work and conclude.

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Deliberation Cycle in BDI-based APLs

Presenting a detailed account of BDI-based agent programming languages is out of the scope of this paper. Moreover, our aim is to provide a general support for extending the plan representation and execution capabilities of a variety of such languages. Therefore we consider a simple model of such languages representing their core components and operations to abstract away from their implementation details or specific features. We call this language RobAPL.

RobAPL has the following components: 1- a belief base and a goal base representing beliefs and goals of the agent at the time, 2- an event base representing goals/events received from the outside world or generated internally by execution of the agent program during the last execution cycle, 3- a plan base containing plans that are being executed by the agent at a time and 4- a rule base containing a set of plan generating rules that are applied to find a suitable plan for achieving a goal or responding to an event.

We treat goals and events uniformly and call them events in the rest of this article. The difference between events and various types of goals in agent programming languages is in semantics of their dynamics. For example a goal of type achievement can be interpreted as a belief state that the agent wish to brings about. In this case, if the execution of a plan for that goal fails, the goal is still in the goal set of the agent and is not removed from the agent goals. Our uniform treatment of event and goals does not put any restriction on semantics of events and goals. To support various semantics for events and goals, we assume that at the beginning of each deliberation cycle, some goals and events from previous deliberation cycles are added to the event base for example because the agent has not yet found a suitable plans for them.

A plan generating rule specifies a partially instantiated plan that can be applied in a certain belief state to reach a goal or to respond to an event. Such a plan is built upon the following basic types of actions.

- Belief-update: updating the belief base. A belief update action has a pre and post conditions. If the pre-condition is entailed by the agent belief base, the belief update action can be executed. The execution alters the belief base such that post-condition of the belief update action is entailed by the belief base.
- Goal-update: updating the goal base by adopting a new goal or dropping an existing one.
- External: performing an external action by invoking a function call. An external action can return a result value.
- Test: performing queries to the agent belief and goal bases to check whether the agent has certain beliefs and goals. If the action succeeds, it bounds the free variables of the queries as result of performing the queries.
- Abstract: performing an abstract action which replace this node with a plan that is associated to the abstract action by a plan generating rule.

RobAPL deliberation cycle consists of a planning and an execution step. In a planning step, plan generating rules are applied to plan for events in the event base. The result of this step is generation of a number of plans added to the plan base. In a execution step, events in the event base are processed again but this time for event-drivel controlling of the plans. Events can be of a type for which a plan needs to be generated, of a type which is used for execution and monitoring of plans or it can be of both types.

RobAPL Plan Overview

A RobAPL plan consists of a hierarchical set of 8 types of nodes. Belief-update, goal-update, external, test and abstract nodes are analogous to belief-update, goal-update, external, test and abstract basic actions. There are also list nodes, resume/pause list nodes and abort/pre-empt list nodes containing other nodes as their children. The root node of each plan is always a list node.

The execution of RobAPL plans (i.e. nodes) is controlled and monitored by a set of conditions on occurrences of events, the agent beliefs and a number of implicit and explicit attributes assigned to nodes. A node's attributes are the following ones among which the ID, priority, estimated execution time and resources are assigned by the programmer.

- ID: is a unique identifies of a node. Each node is uniquely identified by its own name and the name of its ancestors. The name of the list node at the root of a plan is randomly assigned at the run time.
- Status: represents the execution state of a node such as running, finished, etc.
- Outcome: represents the outcome of a node such as success, failed, etc.
- Execution Priority: is an Integer value used for resolving conflicts in parallel execution of nodes.
- Start time: indicates the system time at which a node starts execution.
- End time: indicates the system time at which a node finishes its execution.
- Estimated Execution time: is an estimated amount of overall time required for executing a node.
- Variables: containing all free and bounded variables used by a node.
- Resources: is a set of resource usages of the form $\langle Name, Type, Value \rangle$ where Name is a unique identifier of a resource, Type is one of the *blocking*, *using* and *adding* usage types and Value is an amount of resource usage.

The following are the conditions programmed for each node to control and monitor its execution.

- Start: determines when a node should start executing.
- End: determines when a node should stop executing.
- Invariant: determines when a node should abort executing.
- Pre: is checked right before executing a node and determines whether a node can start executing. If it does not hold, the node finishes its execution with the failure outcome.
- Post: is checked right after a node finishes its execution and determines whether the execution was successful. If it does not hold, the node finishes its execution with the failure outcome.
- Pre-empt: determines when a node should be pre-empted.
- Pause: determines when a node should pause executing.
- Resume: determines when a paused node should resume executing.

- Repeat: determines whether a node should repeat executing.
- Resource: determines when required resources of a node is available.

The start, end, invariant, pre-empt, pause and resume conditions are of the form $\langle Event, Expression \rangle$. Thereby, *Event* denotes an occurrence of an event $e^{[t_s,t_e]}$ and *Expression* is a boolean expression over the content e and the occurrence time interval $[t_e, t_s]$ of event $e^{[t_s,t_e]}$, the node's attributes, the status, outcome, start time and end time of other nodes qualified by their unique IDs and the system time. A condition $\langle Event, Expression \rangle$ holds whenever an event $Event$ occurs and the expression $Expression$ holds true. The event $Event$ can be empty which is defined as an atomic event $empty^{[t,t]}$ occurring at every time t. The pre, post and repeat conditions are logical queries to the agent belief base combined with a boolean expression over the node's attributes, status, outcome, start time and end time of other nodes and the system time. Such queries can only contain anonymous variables. In other words, no variable is bounded as result of these queries. These three types of conditions are only checked once when a node is going to start execution or it finishes its execution. The other conditions are continuously monitored during the time that they are allowed to make transition in a node execution status. An agent has a pool of resources that nodes can query for resource availability. Moreover, the resource pool notifies the availability of resources when a node is waiting to acquire some resources.

RobAPL Operational Semantics

In the execution step of a deliberation cycle, plans are executed by processing all events of the event base in first-come first-served order by so called macro steps. In the beginning of a macro step, an event is processed making some conditions of some nodes in the plan base true. All such nodes make parallel and synchronous atomic transitions referred to as micro step. These transitions alter nodes' attributes which can make other conditions true resulting in another micro step. Micro steps are applied until no more micro step is possible.

The atomic transitions are defined in terms of atomic changes in execution status of individual nodes. At the beginning, all nodes are initialized in the Inactive state except the root node of each plan which is initialized in the Waiting state. In the Inactive state, none of the conditions of a node is monitored. A node in a Waiting state transits to the Executing state whenever its start condition becomes true, its pre-condition holds and its required resources are available. If the pre-condition does not hold, the node transits to the Iteration-Ended state having the Failure outcome. If required resources are not available, the node transits to the Waiting-Resource state from which it transits to the Waiting state again when resources become available. Upon transiting to the Executing state, the action of an action node is executed which succeeds or fails. We assume all actions are performed in a synchronous way. By the synchronous execution we mean the next micro step is performed when all actions of action nodes in the Executing states are finished. For a Long running action which could long delay a micro step, the node can start the action by commanding an external component and then wait for the result to be received as an external event. When the end condition of a node becomes true, an action node transits to Iteration-Ended state and its success or failure is determined by checking its post condition. Then if the repeat condition of the node is evaluated to true, the node repeats its execution by transiting from the Iteration-Ended state to the Waiting state. Otherwise it transits to the Finished state. List nodes act as container of other nodes. After a list node transits to the Executing state, its child nodes transit to the waiting state which are then monitored for execution. When the end condition of a list node becomes true, the list node does not immediately transit to the Iteration-Ended state but to the Finishing state waiting for its children being executed to finish their executions.

A node fails whenever one of its pre, post, invariant or pre-empt conditions is violated. A node also fails if one of its ancestor fails or a pause condition of one of its ancestor evaluates to true. When a failure occurs, action nodes in the Executing state abort their executions, list nodes being executed transit to the Failing state waiting for their children to be aborted and action and list nodes in inactive or waiting states skip execution. The outcome of a node specifies whether the execution of a node for the current iteration was skipped, successful or failure, whether it was failure of the node itself or its ancestors or whether it was due to the preemption of the node or its ancestors.

A node pauses its execution when its pause condition becomes true. The node first fails its children and then goes to the Paused state. When the resume condition of a paused node evaluates to true, it goes to the Resume state waiting for its resume list nodes to finish executing and then transits to the Waiting state. When a node is paused, its children are put in the Inactive state if they were in the Inactive or Waiting state or if their repeat condition evaluates to true. Other children transit and remain in the Finish state.

The execution semantics of resume/pause and abort/preempt list nodes are different than of the other types of nodes. These special types of nodes are for handling cleanup and wind-down activities when other nodes are paused, resumed, failed or pre-empted. The abort/pre-empt and resume/pause nodes transit from the Inactive state to the Waiting state when their ancestors are aborting/pre-empting or resuming/pausing. A list node which is failing/pre-empting or pausing/resuming waits for its abort/pre-empt or resume/pause children nodes to finish their executions before aborting or pausing/resuming its execution.

A difference between RobAPL and *PLEXIL* is the introduction of resume/pause and abort/pre-empt list nodes in RobAPL. The abort, pre-empt and pause list nodes are considered for execution before their parents are failed, preempted or paused. Similarly, resume list nodes are considered for execution before their parents are resumed. This facilitates a structured and bottom-up implementation of clean-up and wind-down activities for nodes that are failed, pre-empted or paused and support performing preresumption tasks before resuming nodes. Another difference is the distinction made between failing and pre-empting

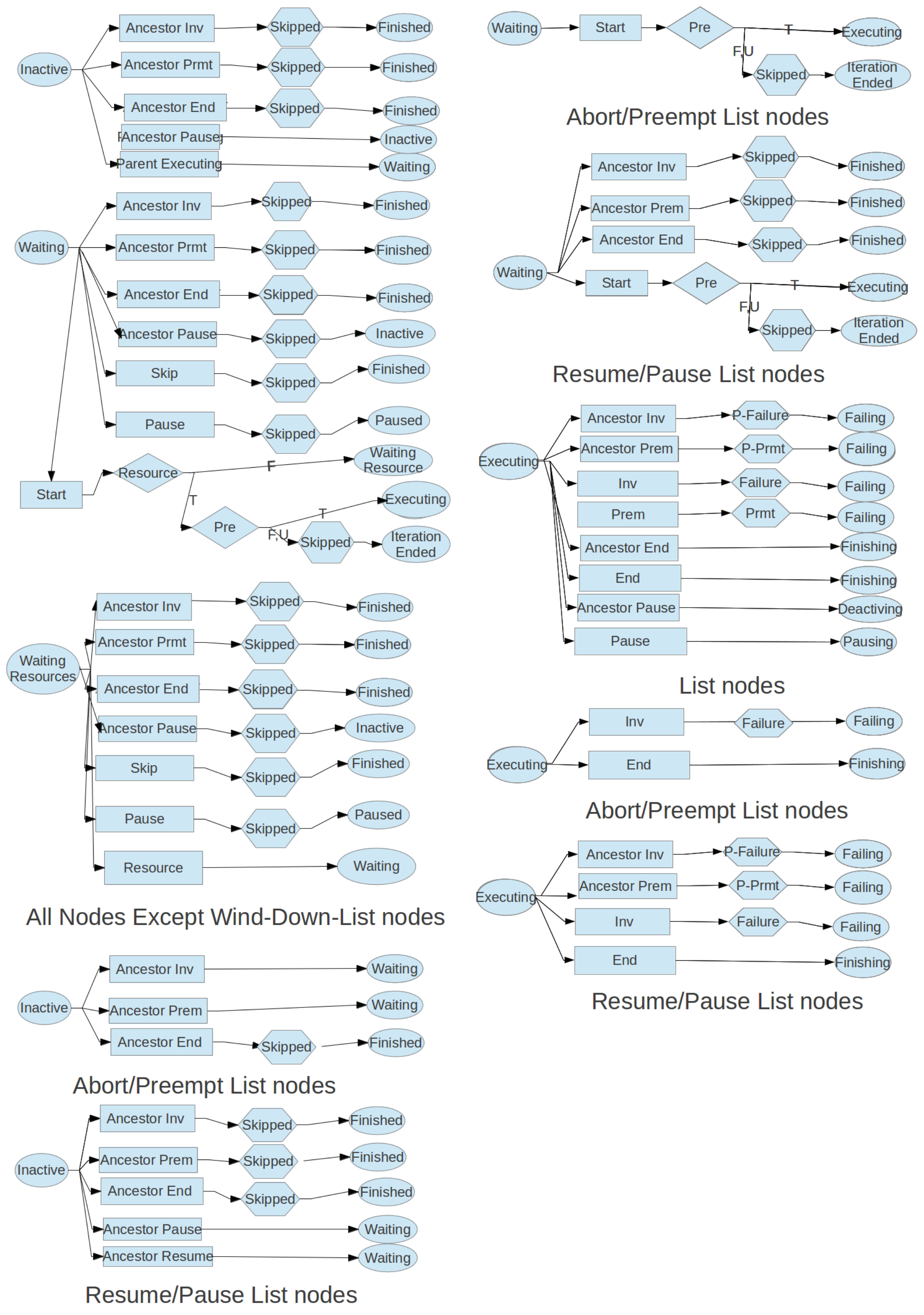

Figure 1: RobAPL atomic transition Diagrams

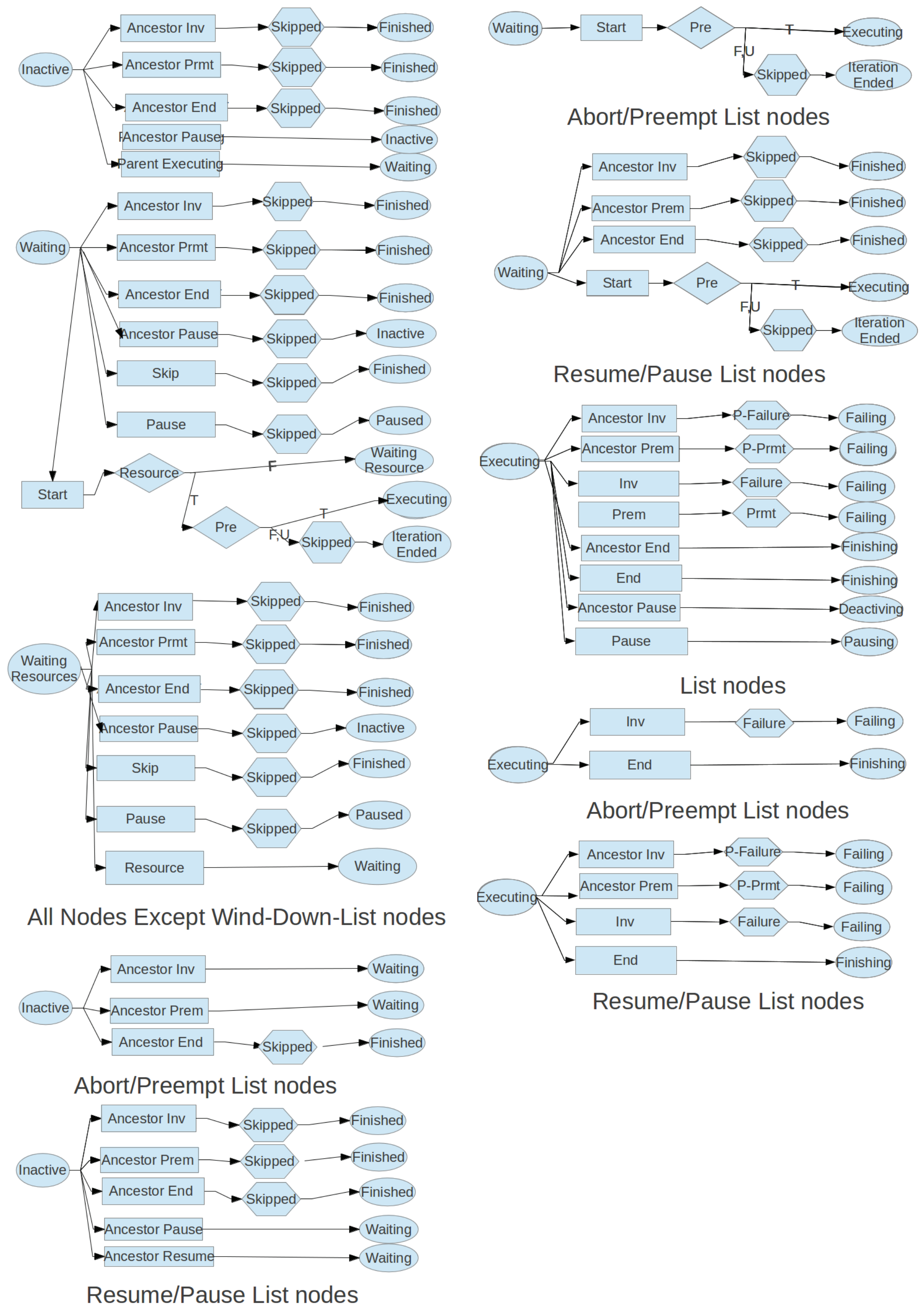

Figure 2: Figure 1 Continued

nodes in RobAPL to distinguish between execution failure, and pre-emption as the result of resource scheduling. This supports utilizing an external scheduler to monitor the plan execution to control pausing or pre-emption of plans based on their deadlines, priorities and available resources.

In each micro step, node transitions are performed in parallel and synchronously. There can be two sources of conflicts in parallel transitions of nodes. One type of conflict is when two nodes require a common resource of which is not enough available to be assigned to both. Similarly, access to shared variables and belief base and goal base needs to be synchronized. For example, two test nodes could attempt to bound a shared variable to two different values. Whenever the execution of two nodes is conflicting, they are executed in the order of their priorities. The other source of conflict is when more than one transition is available for a node. Such conflicts are resolved based on priorities of transitions.

Figure 1 presents semantics of atomic transitions in similar notations to transition diagrams of Plexil (Tara et al. 2006) as follows. The eclipses represent node states. The rectangles represent condition changes that cause a transition from a node state. Only the condition change explicitly represented causes the transition. The diamonds represent checks and the hexagons represents node outcomes. Transitions are represented by directed arrows. If multiple transitions are simultaneously enabled, the top-down order of presenting transitions represent the precedence order. The T, F and U represents the evaluation of a condition to true, false and unknown. The abbreviations Inv, Prmt, P-failure and P-Prmt correspondingly represent Invariant Pre-empt conditions and Parent-Failure and Parent-Pre-empt outcomes.

Conclusion

The paper presents a work toward addressing plan representation and execution control requirements of agent programming languages presented in (Ziafati et al. 2013) to facilitate their use in robot programming. The *PLEXIL* language is adapted to be integrated in the BDI-architecture implemented by BDI-based agent programming languages. This includes introducing execution nodes for querying and manipulating agent's beliefs and goals and presenting a theoretical framework for interleaving the execution of *PLEXIL*-like plans with the plan generating phase of agent programming languages in each deliberation cycle of an agent. The paper also extends the *PLEXIL* language to support pausing, resuming and pre-empting plans and facilitating the implementation of clean-up and wind-down activities when pausing, resuming, pre-empting and aborting plans.

Various future works are foreseen to mature the presented work. The proposed language should be implemented and used in practice to justify its usability for robot programming. Moreover, it is hard to manually verify whether the presented semantics follow the intuitions behind various operations of the language. It is also hard to manually verify whether various determinism properties of *PLEXIL* hold for the RobAPL language. However, similarity of RobAPL syntax and semantics to *PLEXIL* makes it amenable for formal analysis of its properties similar to formal analysis of *PLEXIL*.

Another interesting future work is to look into research in the agent community on life cycle of goals (Thangarajah and Harland 2011) and their various types such as achievement and maintenance. Investigating the relation between semantics of execution, suspension and abortion of goals and semantics of similar states of plans can help to better understand and model the both. Finally, for the better usability, syntax sugars and programming patterns are to be identified to support higher level macros such as IF, For and While statements in the definition of plans.

Acknowledgement

Pouyan Ziafati is supported by the National Research Fund (FNR), Luxembourg.

References

Bordini, R. H.; Braubach, L.; Gomez-sanz, J. J.; Hare, G. O.; Pokahr, A.; and Ricci, A. 2006. A survey of programming languages and platforms for multi-agent systems. *INFORMATICA-...* 30:33–44.

Bratman, M. E. 1999. *Intention, Plans, and Practical Reason.* Cambridge University Press.

Gilles, D.; Cesar, M.; ; and Corina, P. 2008. A small-step semantics of PLEXIL. *NIA Technical Report* (2008-11).

Rao, A. S., and Georgeff, M. P. 1991. Modeling Rational Agents within a BDI-Architecture. In Allen, J.; Fikes, R.; and Sandewall, E., eds., *Proceedings of the 2nd international conference on principles of knowledge representation and reasoning (KR'91)*, 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

Rao, A. S., and Georgeff, M. P. 1995. Bdi agents: From theory to practice. In *Proceedings of the first international conference on multi-agent systems (ICMAS-95)*, 312–319.

Tara, E.; Ari, J.; Corina, P.; Reid, S.; Kam, T.; and Vandi, V. 2006. Plan Execution Interchange Language (PLEXIL). *NASA Technical Memorandum* (TM-2006-213483).

Thangarajah, J., and Harland, J. 2011. Operational behaviour for executing, suspending, and aborting goals in BDI agent systems. *Declarative Agent ....*

Verma, V.; Estlin, T.; Pasareanu, C.; Simmons, R.; and Tso, K. 2005. Plan Execution Interchange Language (PLEXIL) for Executable Plans and Command Sequences. *International symposium on artificial intelligence, robotics and automation in space (iSAIRAS)* 2005(September):5–8.

Ziafati, P.; Dastani, M.; Meyer, J.-J.; and Torre, L. 2013. Agent programming languages requirements for programming autonomous robots. *Programming Multi-Agent Systems* 7837:35–53.