A Rule-Based BDI Agent Architecture to Support Reactive and Proactive Behaviors

Bong-Ki Sohn*, Hak-Joon Kim**, Keon-Myung Lee*

*School of Electric and Computer Engineering, Chungbuk National University and
Advanced Information Technology Research Center(AITrc)
Cheongju, Chungbuk, 361-763, Korea
**Division Of Electronic, Information, Communication Engineering, Howon University
Kunsan, Chonbuk, 573-718, Korea

dobest@aicore.chungbuk.ac.kr

***Abstract*— This paper proposes a rule-based BDI agent architecture to support reactive and proactive behaviors. The proposed architecture represents agent capabilities as *if-then* rules and plans as *ruleplans* which consist of *if-then* rule sets and control flow constructs among the rule sets. *Ruleplans* allow to easily represent and modify plans because their building components are basically *if-then* rules. It also allows to flexibly execute plan because agent can select an executive path as the environment changes. The architecture is composed of the control module which controls agent by executing control rules and the execution modules which process a task by executing task processing rules considering the task execution context. The control module may continuously give execution control to the execution module which is charge of the task to be reactively processed. By maintaining a kind of goal tree, which is a data structure that keeps track of task execution context, the execution module coordinates behaviors in a similar way to BDI model and proactively processes its task. A priority-based forward reasoning method is adopted for decision-making in those modules. The separation of the agent functionality and priority-based decision enable to make simple behavior control mechanism about what task and rule to execute by rule priority. The proposed agent architecture also seamlessly integrates plan generation and its adaptation and thus allows an agent to behave in an adaptive and proactive way by enabling to solve yet unseen problems. This architecture has been partially implemented by employing an extended Jess rule engine and JADE agent model.**

1. INTRODUCTION

The BDI(Belief-Desire-Intention) model is a conceptual agent model that determines what action it would take based on mental attitudes such as beliefs, desires and intentions[1,2]. Beliefs reflect knowledge about the world and desires are possible courses of actions available to the agent. Intentions are the desires that it has committed to bring about. The concept of goals is used instead of desires and plans are predefined as the means with which agents achieve their goals in the BDI agent architectures[3,4,5]. In the previous BDI-based agent architectures, a plan for achieving goal is described as a set of actions that must be ordered. The constraint on plan representation specifying a single execution path reduces expressive power and flexibility of execution of plan. In the previous BDI-based agent architectures, behavior control mechanism is complex because the beliefs, goals, intentions and plans are commonly maintained and manipulated by agent. The previous BDI-based agent architectures also do not support plan generation by planning methods. Therefore, we propose a rule-based BDI agent architecture that supports flexible plan representation, simple behavior control mechanism, and plan generation by planning methods.

This paper is organized as follows: Section 2 introduces a rule-based BDI agent model. Section 3 presents the functional architecture of the proposed agent model. Section 4 describes how to represent the proposed agent knowledge such as rules, goals, and *ruleplans*. Section 5 explains the planning method employed in the proposed agent. In section 6, we sketch how this architecture might be implemented in the existing JADE agent model and Jess rule

engine. In final, Section 7 draws conclusions.

2. RULE-BASED BDI AGENT MODEL

2.1 Rule-Based BDI Model

The BDI model has come to be possibly the best known and best studied model of practical reasoning agents[2]. There are several reasons for its success, but perhaps the most compelling is that the BDI model combines a respectable philosophical model of human practical reasoning[9], a number of implementations, several successful applications, and finally an elegant abstract logical semantics, which have been taken up and elaborated upon widely within the agent research community. The practical reasoning is the process of deciding, moment by moment, which action to perform in the furtherance of our goals. Practical reasoning involves two important processes: deciding what goals we want to achieve, and how we are going to achieve these goals. The former is known as deliberation, the latter as means-ends reasoning[6].

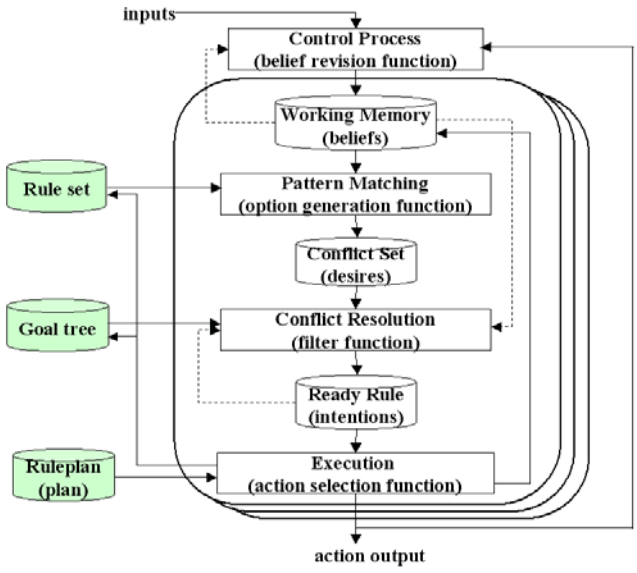

Fig. 1. Rule-based BDI model

In this paper, we introduce a rule-based BDI model for modeling practical reasoning in rule-based system. Figure 1 illustrates the process of practical reasoning in the rule-based BDI agent. There are several main components to a rule-based BDI agent:

- *Control Process* takes perceptual input, interprets it in facts and inserts them into the working memory of an appropriate rule-based system to enable to proceed with its tasks.
- *Working Memory* represents information that the rule-based system has about its current environment such as data related to its task processing and shared data with other rule-based systems.
- *Pattern Matching* process determines executable rules on the basis of its rule set and working memory.
- *Conflict Set* represents the executable rules at the moment.
- *Conflict Resolution* process takes charge of the agent's deliberation process which selects the highest priority rule.
- *Ready Rule* represents the rule to bring about.
- *Execution* process executes the rule.

The *Execution* process may execute a primitive rule that is accomplished at once or a complex rule that takes several reasoning cycles of rule-based system. To complete a complex rule, *Execution* process must refer to *ruleplan* and load appropriate rules into the *Rule set* according to the control flow of the *ruleplan* that is maintained in the Goal tree. Next, *Pattern Matching* and *Conflict Resolution* process are executed, which focuses only on current *Rule set*, *Conflict Set*, and *Ready Rule*. This cycle continues until *Execution* process executes a primitive rule. The process performs the means-ends reasoning as recursively elaborating a hierarchical plan structure, considering and committing to progressively more specific rules, until finally it reaches the *Ready Rule* that is an immediately executable rule.

2.2 Rule-Based BDI Agent Model

A Rule-based BDI agent supports reactive and proactive behaviors by following the rule-based BDI model. It is composed of the control module that decides execution order among tasks and the execution module that selects and executes a rule by considering the task execution context. Both modules embed a rule-based system for decision making and the rule-based systems select tasks and rules to be executed based on priority-based forward reasoning.

Figure 2 represents the rule-based BDI agent model. The control module includes control rules for controlling agent and working memory for reflecting agent's current beliefs about the world. By executing control rules, the control module creates an execution module to process a task for a new event and deletes the execution module that has finished its task. It also forwards events to appropriate execution modules and authorizes an execution module to proceed its task based on execution module's priority. When the shared data is modified, the control module notifies all execution modules of the modification by updating working memory of execution modules.

The execution module executes a rule whenever it is authorized by the control module. Its working memory includes information for processing the given task and shared data. The rule set contains rules dynamically loaded from the rule base by referring to current goal of the Goal tree. It may include primitive rules or complex rules to execute the given task. The execution module makes

decision about what rule to execute based on rule's priority.

The rule-based BDI agent model characterizes functional separation of the agent decision about what task and rule to be executed by rule priority. These features enable the proposed agent model to have simple behavior control mechanism to support reactive and proactive behaviors based on priority of task and rule.

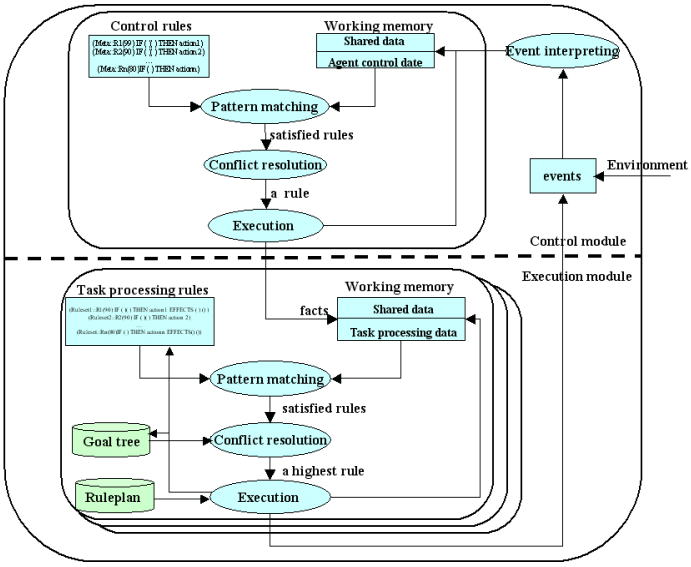

Fig. 2. The rule-based BDI agent model

3. RULE-BASED BDI AGENT ARCHITECTURE

Figure 3 shows the components of the rule-based BDI agent model and their interplay. It consists of five basic components: *Agent Interface module, Agent Controller module, Context Manager module, Subsystem Manager module*, and *knowledge base*. In the following, the individual modules of the system are described in detail.

3.1 Agent Interface Module

This module gathers events from the environment and the agent itself. It also interprets events as facts and forwards interpreted facts to the Agent Controller. Agent Interface processes following events:

- *goal event*, which is a request to accomplish goal.
- *ACL message event*, received FIPA-ACL message from other agents.
- *internal event*, which is generated by interplay among agent components.

The received events are stored in event queue and are interpreted as facts one by one. Agent Interface enables Agent Controller to process its task by inserting the facts into working memory.

3.2 Agent Controller Module

Agent Controller controls agent by executing control rules

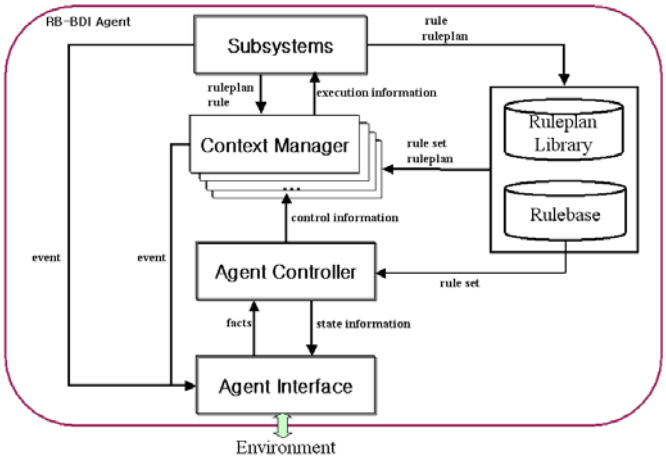

Fig. 3. The rule-based BDI agent architecture

which define following tasks:

- creates Content Manager to independently process a task for a new event.
- deletes Context Manager which has completed its task.
- forwards events to appropriate Context Mangers.
- notifies all Context Managers of the modification of shared data.
- authorizes the Context Manager with highest priority to execute a rule.
- manages priority of Context Manager.

Figure 4 shows a control rule to create Context Manager for processing *maintain* goal event. To keep tracks of particular agent's state, Agent Controller executes *NewMaintainGoalProcessing* rule which creates new Context Manager by calling the internal function *newContextManger*( ).

Agent Controller maintains the priorities of Context Managers. Context Manager has initial priority with control rule's to create itself. As time goes, all Context Manager's priority increases by some degree. When a Context Manager completes a rule, Agent Controller decides which Context Manager to take execution authority based on priority of Context Manager. The Context Manager with highest priority and active state takes a chance to execute a rule. The above control rule has priority 99 which is the highest degree, so the *maintain* goal processing Context Manager takes execution authority to continuously do its task.

```
(ControlRules::NewMaintianGoalProcessing (99)
  IF (event-type goal-event)
    (goal-type maintain)
    (new-event TRUE)
  THEN (perform  newContextManager((ruleset ?x ) (initial-WM  ?y ) (ruleplan ?z)) )
)
```

Fig. 4. An example of control rule

3.3 Context Manager Module

When it is authorized to execute a rule by Agent Controller, the Context Manager proceeds to its task while it maintains task execution context. It independently processes given task against other Context Managers by maintaining Goal tree that includes context of task execution and enables to proactive operation. The Context Manager interacts with subsystems such as Planner and Utility-Maximizer to achieve some goals. It calls subsystems with parameters and receives a *ruleplan* or rule to be executed. To communicate with the Agent Controller, it generates event and forwards it to Agent Interface. The Context Manager generates an event following cases:

- when it has completed its task.
- when it is waiting failure or success response from subsystems or internal functions.
- when it modifies the shared data during its task processing.
- when it has executed a primitive rule.

3.4 Subsystem Manager Module

This module manages subsystems such as Planner and Utility-Maximizer. Planner searches a plan at run time and improves agent knowledge base by adding rule and *ruleplan* translating the plan. Utility-Maximizer finds a rule that has the most utility of some utility function at current state. Subsystem Manager receives requests from Context Manager, executes appropriate subsystem, and returns results of the subsystem to the Context Manager.

3.5 Knowledge Base

Knowledge base consists of Ruleplan Library and Rulebase for agent task processing. Ruleplan Library includes *ruleplan* that describes procedural knowledge for accomplishing complex task. It is referenced to execute the rule which action part contains *ruleplan*. Rulebase includes rules for agent and consists of control rules and task processing rules. Control rules are loaded to Agent Controller and task processing rules are dynamically loaded to Context Manager. *Ruleplans* and rules are created by human expert and also by Planner subsystem.

4. KNOWLEDGE REPRESENTATION

The rule-based BDI agent includes static knowledge such as *ruleplans*, rules and dynamic knowledge such as working memory, conflict set and ready rules, respectively, representing belief, desire and intention of BDI model. In this section, we introduce knowledge representation of the proposed agent architecture.

4.1 Rule Representation

In rule-based BDI agent architecture, a rule defines about what goal to achieve when an agent is situated at some states. It consists of rule head, condition part, action part, and effect part. Rule head consists of rule set name, rule name and priority. Rule set name means rule set that the rule belongs to and Context Manager loads rules that have same rule set name. Rule name is identifier of rule and priority is importance degree of the rule. Condition part specifies situational constraints that must be satisfied for a rule to be applicable. Action part describes goal to achieve. Effect part explicitly specifies agent state changes after a rule executes. It is used to search a plan and a rule with highest utility value. In Figure 5, classify-rule belongs to rule set return-goods and has the highest priority 99. If working memory includes fact “return-reason is defective”, then the rule insert fact “task-type is replace-task”. It also specifies world state as “task-type has some value” after executing.

```
(return-goods::classify-rule (99)
  IF (return-reason defective)
  THEN (assert (task-type replace-task))
  EFFECTS (task-type ?task)
)
```

Fig. 5. A rule example

4.2 Goal Representation

In the proposed architecture, six different kinds of goals can be distinguished: *assert*, *retract*, *achieve*, *perform*, *maintain*, *utility-maximize* goal. The *assert* adds facts into and the *retract* deletes facts from working memory. The *achieve* goal just defines desired target states without specifying how to reach it. This goal is achieved by executing *ruleplan* returned by Planner subsystem. The *perform* goal directly specifies the rule or *ruleplan* or internal function to execute. For goal of kind *maintain*, an agent keeps track of the state, and will continuously execute appropriate *ruleplan* to re-establish the target state whenever needed. *utility-maximize* goal specifies the utility function that at current states, selects a rule with highest utility function value. This goal is achieved by Utility-Maximizer subsystem.

assert, *retract* and *perform* goal which specifies internal function or rule are completed within one cycle of rule-based system. However, *achieve*, *maintain*, *utility-maximize*, and *perform* goal which specifies *ruleplan* are completed with several cycles.

4.3 Ruleplan Representation

Ruleplans are the means to achieve the goals such as *perform*, *achieve*, *maintain*, and *utility-maximize*. A *ruleplan* defines a procedural specification for accomplishing complex task. A *ruleplan* consists of Tasks containing several rules which model subtask of complex task and control flow among Tasks. A Task contains Task rule set,

Task functions, Task data. Task rule set includes rules to be used to complete the Task. Task functions and Task data are accessed by the Task rule set.

Control flow specifies execution orders among Tasks, which may have *sequence*, *conditional*, *switch*, *while*, and *goto* flows. Context Manager processes *ruleplan* according to control flow, so it is possible to proactively operate in the rule-based system. Figure 6 shows return-goods *ruleplan* which contains four Tasks and control flow among them. Return-goods task is divided into classification, refunding, replacement, and completion Task. Figure 7 shows control flow of the return-goods *ruleplan.*

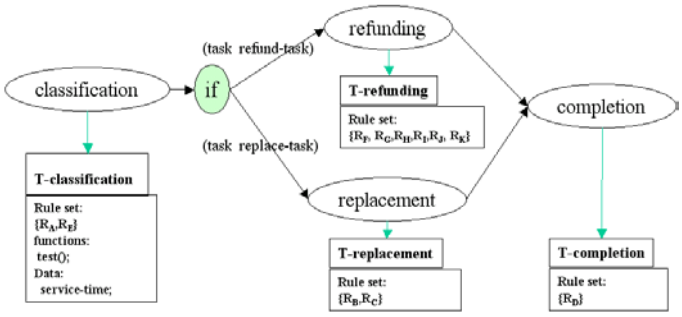

Fig. 6. return-goods *ruleplan*

First at all, the Context Manager executes classification Task according to details of customer's request. After completing the classification Task, if working memory includes fact "task is refund-task", it executes refunding Task and completion Task in sequence.

```
return-goods
{
  classification;
  if(task refund-task)
      refunding;
  else
      replacement;
  completion;
}
```

Fig. 7. Control flow of return-goods *ruleplan*

Context Manager executes *ruleplan* according to its control flow, which is maintained in the Goal tree. Figure 8 shows the *ruleplan* execution algorithm of the Context Manager.

5. PLANNING IN THE RULE-BASED BDI AGENT

The Context Manager accomplishes *achieve* goal by planning mechanism. To do this, *if-then* rules are translated into STRIPS operators. Table 1 shows mapping relationship between if-then rule and STRIPS operator[10]. The *achieve* goal's target states and working memory facts of the Context Manager respectively correspond to Planner's target states

```
Algorithm ExecuteRuleplan(a-ruleplan)
  control-flow:=RLibrary.reference(a-ruleplan);
REPEAT UNTIL END(control-flow)
  a-task := dispatch-task(control-flow);
  focused-ruleset := focus(a-task);
  update-goal-tree(focused-ruleset, a-task);
  rules :=pattern-match(CMworking-memory, focused-ruleset);
  add-task-conflict-set(task-conflict-set, rules);
  REPEAT UNTIL END(task-conflict-set)
    a-rule:=select(task-conflict-set);
    execute(a-rule);
    rules :=pattern-match(CMworking-memory, focused-ruleset);
    add-task-conflict-set(task-conflict-set, rules);
  END-REPEAT
END-REPEAT
```

Fig. 8. *Ruleplan* execution algorithm of the Context Manager

and initial states. Figure 9 shows an example of mapping between rule and operator. Left side represents *if-then* rule and right side is the translated operator according to mapping relationship of Table 1.

```
(RETURN-GOODS ::classify-rule (99)
   IF (return-reason  defective )
     (return-type product)
   THEN (assert  (task-type  replace-task))
   EFFECTS (task-type  ?task)
)
```

```
(return-reason  defective ) (return-type product)
   assert(task-type  replace-task)
(task-type  ?task)
```

Fig. 9. An example of mapping between rule and operator

The Planner subsystem searches plans based on translated facts and randomly selects one. The selected plan are retranslated into a *ruleplan* and returned to the Context Manager. The planner creates a new rule that describes the *ruleplan* in the action part and adds it to rule base. It also adds the *ruleplan* to Ruleplan Library to reuse in the future. The proposed agent can improve its capability since Planner subsystem adds *ruleplan* and rule to agent knowledge base.

Table 1. Mapping relationship of rule and operator

If-then rule	STRIPS operator
condition part	preconditions
action part	action
effect part	postconditions

6. PROTOTYPING WITH JADE AND JESS

The feasibility of the proposed agent architecture has been being evaluated by prototyping with JADE[7] and Jess[8]. JADE is a FIPA compliant software framework for developing agent applications of inter-operable, intelligent, multi-agent systems. JADE is a kind of agent platform middleware and development framework supporting distributed, interoperable multi-agents together with the

corresponding communications infrastructure including the FIPA Agent Communication Language(ACL).

An agent in the JADE sense uses the JADE Agent abstraction and models an agent's tasks via the JADE Behaviors abstraction where each agent may dynamically instantiate its own behaviors as needed. Multiple behaviors belonging to the agent may execute concurrently using a round-robin, non-preemptive policy. The Java language is used to implement the agent infrastructure and base classes for agents and behaviors. However, there are no limitations for the kinds of software systems which can be embedded in the agents and behaviors.

The proposed agent conceptually is a JADE agent initially having Agent Interface Behavior and Agent Controller Behavior. These Behaviors are CyclicBeahviours since they must execute continuously during agent's life. Agent Interface Behavior continuously accepts events, interprets it in facts, and forwards facts to Agent Controller. Agent Controller Behavior contains a Jess rule engine which is well known forward reasoning system written in Java and continuously control rules. It also dynamically creates and deletes Context Manager Behaviors embedding a Jess rule engine. Context Manager's decision making is done by Jess rule engine. By maintaining Goal tree, the Context Manager makes proactive task processing cooperating with Jess rule engine.

7. CONCLUSIONS

In this paper, we proposed a rule-based BDI agent architecture that supports flexible plan representation, simple agent control architecture, and plan generation by planning mechanism. The proposed agent architecture has the following features:

First, it provides flexibility of plan construction and execution by representing plan as structured rule sets called *ruleplan*. It is possible to proactively process the given task by referring to the control flow of *ruleplan.*

Second, the proposed agent architecture decides about what task to do among tasks and what rule to execute taking into account the context of the selected task based on the priority of tasks and rules. Decision making of rule-based BDI agent is based on priority-based forward reasoning. Priority-based decision enables to make simple decision about what task and rule to execute by priority and to reactively and proactively operate by having a task continuously take highest priority.

Third, it is possible to automatically generate *ruleplans* with the help of the Planner subsystem. To do this, planner subsystem translates *if-then* rules into STRIPS operators to search plans and generates plans for achieving target states. A generated plan is retranslated into a *ruleplan* and is inserted to agent knowledge base for reuse in the future.

By the *ruleplan* and priority-based decision, the rule-based BDI agent can reactively and proactively execute various tasks by simple behavior control mechanism. It also incrementally improves agent capability through planning mechanism.

The proposed agent architecture can be applied to electronic commerce agents and workflow agents which use procedural knowledge to solve problems and require rapid agent prototyping and easy modification of agent knowledge in the dynamic environment.

ACKNOWLEDGEMENTS

This work was conducted with the support of the Korea Science and Engineering Foundation(KOSEF) via the Advanced Information Technology Research Center(AITrc).

REFERENCES

[1] M. Georgeff, B. Pell, M. Pollack, M. Tambe, and M. Wooldridge, "The Belief-Desire-Intention Model of Agency," *in ATAL*, 1999.

[2] A. Rao, M. Georgeff, "BDI Agents: From Theory to Practice," *Proceedings of the First International Conference on Multi-Agent System-ICMAS95*, San Francisco, USA, 1995.

[3] A. Pokahr, L. Braubach, and W. Lamersdorf, "Jadex: Implementing a BDI-Infrastructure for JADE Agents," *in: EXP - In Search of Innovation (Special Issue on JADE)*, Vol 3, No. 3, pp.76-85, Telecom Italia Lab, Turin, Italy, September 2003,

[4] N. Howden, R. Ronnquist, A. Hodgson, and A. Lucas, "JACK Intelligent Agents - Summary of an Agent Infrastructure," *Proceedings of the 5th ACM International Conference on Autonomous Agents*, Canada, 2001.

[5] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldrige, "A Formal Specification of dMARS," *INTELLIGENT AGENT IV: Agent Theories, Architectures, and Languages*, M. Singh, M. Wooldrige, and A. Rao(editors), LNAI 1365, Springer-Verlag, 1998.

[6] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, 1998.

[7] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "JADE-A FIPA-Compliant Agent Framework," *Proceedings of 4th International Conference and Exhibition on The Practical Application of Intelligent Agent and Multi-Agents*, pp.97-108, London, UK, 1999.

[8] E. Frieman-Hill, *Jess in Action: Java Rule-Based Systems*, Manning Publications Company, 2003.

[9] M. E. Bratman, D. J. Israel, and M. E. Pollack, "Plans and Resource-Bounded Practical Reasoning," *Computational Intelligence*, Vol. 4, pp. 349-355, 1988.

[10] R. Fikes, and N. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 2, pp.189-208, 1971.