2013

Measuring the reactivity of intelligent agent programs

Tiancheng Zhang
University of Wollongong

Recommended Citation

Zhang, Tiancheng, Measuring the reactivity of intelligent agent programs, Master of Computer Science thesis, School of Computer Science and Software Engineering, University of Wollongong, 2013. https://ro.uow.edu.au/theses/4587

Research Thesis

Measuring the reactivity of intelligent agent programs

UNIVERSITY OF WOLLONGONG

Tiancheng Zhang

School of Computer Science and Software Engineering

University of Wollongong

A thesis submitted for the degree of

Master of Computer Science

March 2013

Acknowledgements

I want to express my sincerest appreciation to my supervisor, Professor Aditya Ghose, who continually and constantly conveyed a spirit of adventure in my research. When I felt frustrated, it is him that encouraged me and helped to overcome various barriers. In addition, he spent a lot of time on me, even on weekend and public holidays. This thesis might not have been completed without his persistent guidance as well as assistance.

I would like to thank my co-supervisor, Doctor Hoa Khanh Dam, who introduced agent technology to me and guided me in figuring out measures for agent reactivity. Without him, I may not even know what agent technology is.

In addition, I would also like to thank all authors of papers I referenced or read. They helped me to gain a deep understanding of my research area.

Abstract

The booming of intelligent agent technology over past few decades brings a surging number of agent applications in various areas. There also have a large number of designs as well as programming languages been introduced in the literature in the agent oriented area. However, very little work has been dedicated to define quality measures for the development of an agent-based system. Previous efforts mostly focus on adopting classical measures such as using coupling (degree of program dependency) and cohesion (degree of function relationship in single module) to measure the quality of agent programs. I maintain that its time to work on a set of software quality measures that are specific to the distinct characteristics of agent-based systems. In this thesis, two methods are purposed to measure the reactivity of agent systems, which provide indications on how agent systems respond to changes in their environment in a timely fashion. A prototype tool is developed integrated with Jason, a well-known agent-oriented programming platform, to calculate reactivity of each agent in agent system based on their plan libraries, and conducted several experiments to demonstrate the reliability of reactivity measure. In addition, an agent behavioural profile is introduced which is an overview of relationships of actions in agent plan library. Based on agent behavioural profile, definitions of agent behavioral profile identity, entailment as well as preservation were proposed, which ensure original agent's behaviours could be preserved while performing reactivity enhancement.

Contents

List of Figures

literature review is conducted to give a background knowledge of BDI agent, Jason platform, agent reactivity and behavioural profile. In chapter 3, a discussion is made on different factors that contribute to the reactivity of an agent program. In addition, two reactivity measures and the algorithm to calculate them are given. In order to prevent agent from losing its original behaviour, several definitions for agent behavioural preservation are introduced. In chapter 4, a tool which supports agent developers to measure the reactivity of their agent programs is demonstrated and results from several experiments are analysed. In chapter 5, a conclusion is made and some future work is outlined.

Chapter 2

Background

2.1 Agent Technology

According to Wooldridge [52], agents have two significant properties. First, agents can perform some automatic actions to some extends. That means they can have their own decision on what to perform for the purpose of achieving a goal, instead of receiving command and controlled by other entities. The second is they are able to interact and communicate with other agents. The interaction mentioned above is not simply sending and receiving messages. In addition, they can participate in similar kind of social activity that human beings engage in daily life. For instance, they can work together, get themself organised, argue as well as compromise between each of them. Agent system is a computer system can perform independent action as role of its user or owner. The difference between an object and an agent is that the object performs tasks which are invoked by other objects. This may cause deadlock. Whereas the agents, can react according to the environment. Therefore, agents have little or no connection between them and less deadlock issue needs to be considered. There is a slogan: objects perform actions for free; agents perform actions because they want to.

2.2 BDI agents

There are many different types of studies, which have been conducted as foundation for programming agent-oriented system, such as plans and resouce-bounded reasoning [12], rationality reasoning [20]. In addition, a number of intelligent agent architectures such as logic-based architecuture, reactive architecture and layered architecture were created [53]. Belief-Desire-Intention (BDI) is a model that is one of the most mature and commonly used. The BDI agent model is developed for intelligent agent to solve particular problem in agent programming. It separates the process of choosing plans and executing plans in order to balance time spent on both processes [45].

- Belief is the knowledge of an agent to the world including the knowledge of itself and other agents. It is the information state of an agent.
- Desire is goals or objectives that an agent plans to complete. It is the motivation state of an agent.
- Intention is a committed option of an agent. It is the deliberative state of an agent which is stronger than desire.

Regarding to plan library for BDI agent, it consists of original goal, condition gateway as well as several plan bodies. Each plan is made up of an event that can start the plan, a decision gateway and several steps of the plan in the format of $E : C \leftarrow B$. A triggering event E can be a goal, a task, an internal or external event. The context condition C plays a role of a gate. Therefore, only when the decision gateway returns true, can process goes to plan steps. Plan steps B can have a set of simple actions as well as subgoals. A subgoal can invoke another plan in the plan library.

Based on the BDI execution cycle algorithm [10], a figure 2.1 can be used to illustrate the lifecyle of BDI agent. The BDI execution cycle can be summarised in following steps:

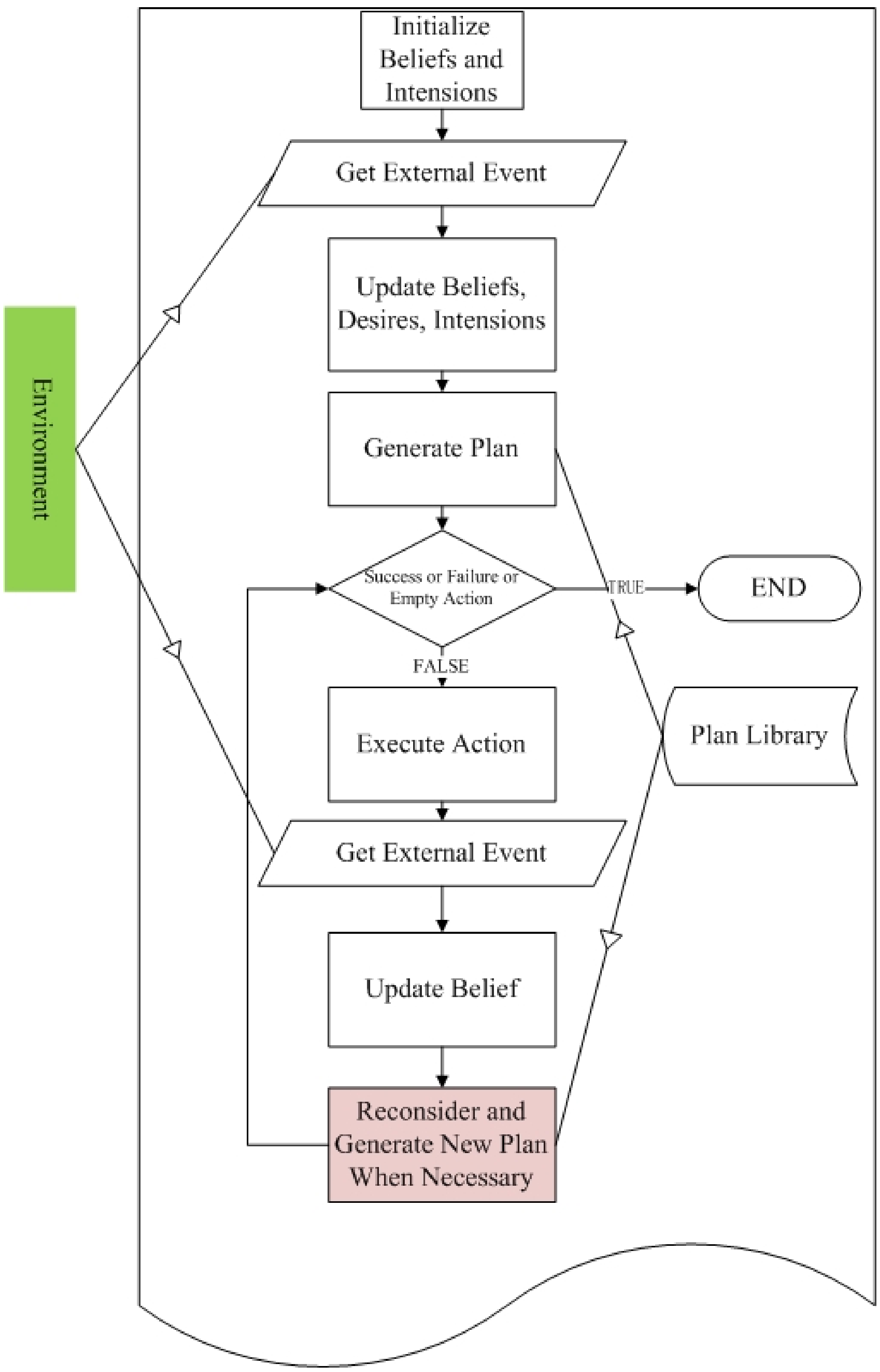

Initialize Beliefs and Intensions
Get External Event
Update Beliefs, Desires, Intensions
Generate Plan
Success or Failure or Empty Action
TRUE
END
FALSE
Execute Action
Plan Library
Get External Event
Update Belief
Reconsider and Generate New Plan When Necessary
Environment

Figure 2.1: BDI execution cycle

1. Agent initializes its pre-defined belief and intentions. Pre-defined beliefs and intentions are usually placed at the top of the plan library.
2. Agent gets next perception from the environment through its sensors. It is a preparation for updating agent's belief.
3. Agent updates its beliefs, desires and intentions in sequence. At this stage, agent can make a decision and commit the decision based on latest knowledge of the environment.
4. Base on current status, agent generates an available plan to achieve the goal.
5. When the plan is not finished or not empty, agent executes the action which is placed at the top of the plan and removes the processed action in the plan.
6. Agent then tries to get next percept and update its belief.
7. Based on current status, agent makes a decision on whether it is necessary to reconsider intention and generate a new plan.
8. Agent then repeats steps 5-7 until the plan accomplishes, fails or is empty.

When a plan failed in step 5 or 8, agent tries another plan in the plan library until there is no candidate plan exists. What worth mentioning is any failures in plan can cascade to upper nodes in events planning tree. That is, a child plan failure can trigger a parent plan reconsideration. If a plan is successfully executed, agent stops current plan. However, agent still keep alive and wait for other goals, tasks or events.

P1 $te : true \leftarrow a_1; !g; a_2$
P2 $g : c1 \leftarrow a_3$

When an event te is generated and the agent decides to handle this event, it commits to execute a plan ($P1$ in this example) by pursuing a set of actions as defined in the body of plan $P1$. It means that the agent firstly executes action a_1 and then generates an internal event $!g$ (representing the need to achieve goal g).

The agent then suspends its current course of action (i.e. intention) and tries to achieve g. In this case, let assume that $c1$ holds agent's current beliefs. The agent then commits to execute plan $P2$ and consequently performs action a_3. After the agent finishes executing action a_4, i.e. successfully achieves goal g, it will resume to pursuing the course of action in $P1$, i.e. performing action a_2.

In step 7, agent can only reconsider the current plan and generate new plan in goal-oriented behaviour when there is a context condition before next action. In the example above, plan reconsideration can only be performed in context condition c1. Therefore, action a_3 can only be executed when context condition c1 is true. However, action a_2 is performed regardless of the context condition c1.

2.3 Jason

A lot of platforms can be used to implement agent systems based on BDI model such as Jason [10], JACK [14], Jadex [42], PRS [31], dMARS [19], or 3APL [26] (refer to [7, 8] for more agent platforms). Jason is an agent programming environment that supports extended AgentSpeak. The essential part of AgentSpeak programming is to define plan libraries for agents. Therefore, agents can possess knowledge of how to handle events from environment. Jason makes it possible for agents to communicate and cooperate with one another in a high-level way due to its rich environment. It's communication concentrates on knowledge level. In other words, communication among agent is based on belief, goal and intention. In Jason, an interpreter executes multi-agent systems based on reasoning cycles which can be divided into 10 key steps [10]:

1. Perceiving changes from environment that agent resides
2. Updating the belief of agent
3. Receiving messages or commands from other agents
4. Selecting messages that can be accepted by agent
5. Selecting an event from events pool

6. Fetching all relevant plans for the selected event
7. Making decision on the applicable plans
8. Filtering one applicable plan from all candidate plans
9. Selecting an intention from intention queue for performing that intention
10. Performing one action of the selected intention

As an implementation of BDI architecture, Jason can handle failures such as test goal failure, action failure and no applicable plans for triggering events. Currently, three distinct execution modes are provided by Jason:

- Asynchronous Mode: By default, all agents in a multi-agent system run asynchronously. They do not wait for each other when they finish their own execution cycle.
- Synchronous Mode: All agents perform reasoning cycle together. If one agent finishes it's execution cycle first, it will wait until all agents in the system finish their reasoning cycle execution.
- Debugging Mode: The debugging mode is similar to synchronous mode. However, instead of starting next reasoning cycle automatically, user has to click "next" to perform the next reasoning cycle.

2.4 Reactive Agent Programming

For agents that make decisions instantly rather than based on experience, those agents are pure reactive agents. Once pure reactive agents receive changes from the environment, they will make corresponding reaction based on the predefined plan libraries [54]. Plans in pure reactive agent are in the format of **perception $\rightarrow$ actions**.

A programming system based on Open Constraint Programming(OCP) framework was designed to implement web application reared by reactive agent. The system is implemented in Constraint Logic Programming Language. Reactive behaviours can be triggered if some agents change their stored constraints that other agents react against. OCP reactive system architecture contains four layers, namely constraint store, registry, web server and user interface [56].

A constraint and objective based reactive BDI agent language CASO (Constraint AgentSpeak(L) with Objective) was introduced by Dasgupta and Ghose. CASO has many advantages over AgentSpeak(L). It allows people to quantitatively express goals of agents. In addition, expressive abilities can be enhanced by integrated constraints. The introduction of constraints helps agent seeking optimized solution at any time. CASO enables agent to select plans in an efficient way with parameter foresee techniques [18].

2.5 Measuring Reactivity

Due to the fact that an agent is inhabited in the environment, it can react to updates happen in the enviroment. In figure 2.2, sensors installed on agent can constantly monitor changes from the environment. Sensors then convert signals from the environment to agent understandable digital message. There are many different kinds of physical sensors such as sound sensor, light sensor, shock sensor, etc. Apart from physical sensors, software sensors are used in agents as well. Upon receiving messages from sensors, agent can make decisions and perform corresponding actions which may in turn, affecting the environment. When agent makes a correct response, this agent is reactive to this particular environment status. Otherwise, the agent fails to reactive to the environment. Due to the uncertainty and variation of environment, research have been conducted on agent reactivity.

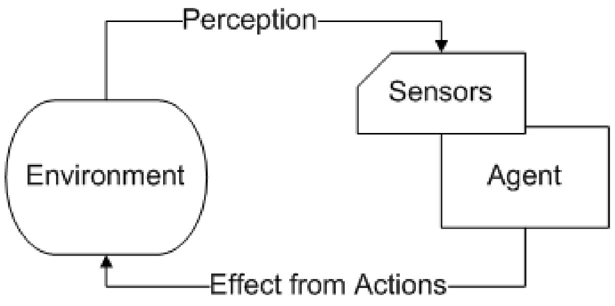


Figure 2.2: Interactions between agent and environment

According to Cernuzzi and Rossi, whether agents can make response to updates that happen in an environment instantly can be used to check reactivity of an agent design model. Based on their case study, BDI and MAS-CommonKADS architectures are reactive [15].

So and Sonenberg introduced definition of proactive agent behaviour. They briefly mentioned that reactive agent behaviour are associated with final conclusion of inference network. When all beliefs in subset have been asserted and those beliefs are true, actions are reactive behaviours [47].

Methodology for building an environment based reactive multi-agent system is given in four main steps by Simonin and Gechter [46]. Firstly, it defines the environment structure and dynamic rules. Secondly, it defines agent perception which allows agents to percept changes in the environment. Thirdly, it defines agents' interaction mechanisms in both local and cooperative level. The last step is to measure results as an emergent structure. For measure the static problem without constraints change, equilibrium of the system is achieved when all agents stop interacting with the environment. Whereas for the dynamic problem with constantly changed constraints, a measure for organization has to be designed. Two methods of other researchers are introduced to measure organization. However, both methods cannot handle the nature of the native mechanisms architecture. This paper introduces a method to architect agents system that can react to the environment for purposes of attacking problems. However, well studied method of measuring dynamic problem is not given and measure of evaluating reactivity of agents is not introduced.

Same as previous paper, a formal method is introduced by Bounabat et. al [11] to describe and validate a reactive system. In system description, statechat in the format of "?event[condition]/!action" is used to describe the system and can be automatically transformed into a synchronous language ESTEREL. ESTEREL can then be compiled to check temporal properties formally and perform automatic verification. Despite the fact that the method can validate synchronization of multi-agent system, no formula has been given to measure agent reactivity.

2.6 Other agent metrics

A discrete evaluation method is proposed by Cernuzzi and Rossi to evaluate pro-activeness for both BDI agent modelling technique [33] and MAS-CommonKADS modelling method [30]. They argued that in BDI, pro-activeness is partially covered in plan model. However, it is impossible to specify how to assume different objectives dynamically.

Agent proactiveness and reactiveness are analysed by Lin and Carley. They argued that agent style only influences performance of organization which is under moderate time pressure. In addition, training type and internal condition can influence the effect of agent style [35].

So and Sonenberg put forward a definition of proactive agent behaviour. They argued that proactive agent behaviours are actions related to intermediate conclusion from any given inference network. Level of pro-activeness depends on the number of prerequisite beliefs [47].

Alonso et. al. [3] purposed a measure to evaluate proactiveness of agent. Based on research and experience, they identified that properties of initiative, interaction and reaction were relevant to agents proactivity. Three measures on roles count, goals count and events to complete goals are designed. Measures for calcuating methods number in class and category of messages are figured out for interaction. Handled requests count and agent procedure complexity measures are developed to evaluate reaction. Based on all single values of each attribute and their weights from experience, a single value for proactivity from 0 to 1 can be calculated.

Cernuzzi and Rossi argued that social ability of agent models can be evaluated by following properties. They are organizational relationships, interaction with agent, types of interaction, commitments, conversations with agents as well as interfaces with other entities [15].

Alonso et. al. [1] purposed a measure to evaluate social ability of agent. They concluded that communication, opperation as well as negotiation can affect agents social ability. For each of these attributes, measures based on different elements are created. For communication, measures of event response time, events size average, received events and sent events are developed. For cooperation, measures of rejected request from other agents and published agent functions are designed. When it comes to negotiation, measures such as achieved goals for agents, number of events sent by called services and events triggered to ask for a service are introduced. By applying weighting technique, weighted average from 0 to 1 on social ability can be gained for each agent as well as for the whole system.

In order to evaluate agent design models, Cernuzzi and Rossi purposed that agent autonomy can be evaluated by whether agents have control on both internal state and their own behaviour. They concluded that BDI and MAS-CommonKADS agent models are autonomous [15].

A goal-focused autonomy assessment is introduced by Barber and Martin [36]. They argued that autonomy can be described in Sensible Agent-based system using the tuple (G, D, C), where G is the goals that agents are making decisions on, D represents what measures agent will take to pursue goals defined in G, C is authority constraint which will guarantee agents can make decisions of the decision-making group. The autonomy value comes from the measure is from 0 and 1. As increasement of autonomy value, agent has more control on decision making by itself.

Autonomy measure based on agents social integrity and social dependence was introduced by Huber [28]. Measure for social integrity is a function that calculates a minimum value from structure measure. The structure measure calculates a minimum value from all possible influences paths for that internal structure. When it comes to the social dependence, it can be calculated by totalling tasks imposed on agent from higher level agents, acceptable tasks from peer agents, contracted tasks with peers and tasks imposed upon lower level agents. By applying a weighting coefficient for both social integrity and social dependence, an average value of agent autonomy can be calculated.

In 2009, Alonso et. al. [2] put forward a measure to evaluate autonomy of agents. From their experience and literature survey, they made a conclusion that agent autonomy is affected by self-motivation, independ working as well as self-learning. For each of these attributes, measures based on different elements were designed. For self-control, measures of complication of structure, status queue size in side agent, complication of agent behaviour are designed. For functional independence, a single measure on rate of executable messages is used. For evolution capability, measures on ability to refresh status, rate of state refresh speed are worked out. By applying arithmetic mean of all values, autonomy of the system can be calculated.

Apart from measuring main agent properties, plan coverage and overlap are also important for agents. Plan coverage illustrates how a goal is covered by plans. Thangarajah et. al. argued that coverage measure of a plan decreases as the increases of sub-goals. Therefore, a plan without sub-goals has a coverage of 1. Plan overlap indicates whether there are multiple applicable plans for a goal. When a goal is not full coverage, overlap measure can be influenced by the structure of goal-plan hierarchy tree as well as the distribution of the overlap in the tree. Minimum value of measure on plan overlap is 0 [49].

2.7 Similar measures in other parts of software engineering

In real-time reactive system, Zheng and Ormandjieva purposed a discrete time Markov chains based measure to estimate the reliability of the system. Initially, each reactive unit gets a Markov model. Then probability of states transition can be calculated using $P = 1 - (1 - P\{l_1\}) \times ...(1 - P\{l_n\})$ where $\{l_1, ...l_n\}$ are multiple transitions from one state to another. The reliability of each subsystem is the sum of differences between the level of uncertainty in its Markov Model and the level of uncertainty of each reactive object. Reliablility of the whole system is the minimum value among subsystems due to the safety-critical character of most real-time system [55].

In 1974, Wolverton used line of codes(LOC) to measure software complexity [27]. LOC metrics is calculated as the total number of source lines of code without blank and comment lines [34]. In 1976, McCabe [38] introduced a widely accepted measure for complexity v = e - n + 2p based on program control graph. In the measure, e represents a sum of edges, n is a sum of nodes and p is a count of linked elements in the graph. To simplify the complexity of calculation, the complexity measure can be expressed as $v = \pi + 1$ when p = 1, where π is the number of predicates in code.

In 1977, Halstead [23] developed several metrics stem from source code properties such as sum of different operators (η_1) and operands (η_2) as well as sum of operators (N_1) and operands (N_2). With these properties, vocabulary size can be defined as $\eta = \eta_1 + \eta_2$ and program length can be calculated by $N = N_1 + N_2$. It is also possible to estimate length of program by $\hat{N} = \eta_1 log_2 \eta_1 + \eta_2 log_2 \eta_2$. Both program volume (V) and difficulty (D) can increase the effort in doing software development, thus E = D * V. Program volume and difficulty can be defined as $V = N * log_2 \eta$ and $D = \frac{\eta_1}{2} * \frac{N_2}{\eta_2}$ respectively. The effort value is hard to understand, so Halstead provided a metric, $T = \frac{E}{\beta}$, to translate effort into the time required for programming. β is usually set to 18 based on Halsteads experiment. Program level can be defined as $L = V^*/V$ where V^* is the estimated volume which can be calculated using metric $V^* = (2 + \eta_2^*) log_2 (2 + \eta_2^*)$. η_2^2 in the metric represents the required input and out parameters. Program level is from zero to one. If the level is closer to one, it means that program is written in the highest level with minimum size. If a program requires more effort, more bugs might occur in delivered product. An estimation on number of delivered bug can be done by $B = \frac{E^{\frac{2}{3}}}{3000}$.

As both McCabe's and Halstead's metrics could not distinguish the additional complexity of nesting code block. Harrison and Magel worked out an adjusted complexity measure in 1981[24]. They introduced the concept of raw complexity value which can be assigned to each node in control flow graph. Following steps might be involved in calculating the total programs' complexity:

1. Determine sub-graph of each selection node. A selection node in the control flow graph has out-degree greater than one.

2. Calculate the adjusted complexity for each selection node by summing up raw complexity of all nodes in the sub-graph including the selection node. For nodes other than the selection node, they have the adjusted complexity equals to raw complexity.

3. Sum up adjusted complexity of all nodes in control flow diagram and get the program complexity.

In 1982, Piwowarski argued that cyclomatic complexity as well as many other refinements failed to distinguish complexity in structured and unstructured programs, nested and sequential control structure and program with case statements [41]. According to his research, he found that unstructured program is more complex than structured one, sequential control structure is easier to understand than nested control and N-way case statement is more efficient than N-1 nested IF statement. He then put forward a complexity measure $N = V * (G) + \sum_i P(i)$, where V*(G) is the adjusted cyclomatic complexity that treat case structure as one predicate and P(i) is the nesting depth of the i^{th} predicate based on Harrison and Magels sub-graph concept [24] mentioned in previous paragraph.

In addition, Kim et. al. [32] designed three entropy concept based complexity measures for object oriented design in 1995. Entropy theory can be used to describe the degree of disorderliness. In measuring class complexity, they applied reference probability of all nodes in Data and Function Relationship (DFR) graph to entropy function. Despite the fact that same reference probability formula is used in inter-object complexity calculation, Object Relationship (OR) graph was used as a replacement of DFR graph. In order to know the complexity of the object oriented program, total complexity was designed based on class and inter-object complexity. Total complexity is the sum of all class complexity plus the result of a constant multiplied by inter-object complexity of the program.

Three quality metrics on Object Orient Design were introduced by Martin in 1994 [4]. Afferent couplings(Ca) can be measured by the number of exterior classes that depend on interior classes of this category. While efferent couplings(Ce) can be calculated by the sum of interior classes that depend on exterior classes of this category. Afferent couplings and efferent couplings can then be used to calculate Instability (Ce / (Ca + Ce)) which is from 0 to 1. 0 stands for maximum stability while 1 stands for maximum instability.

Husein and Oxley introduced a tool CCMETRICS to get software coupling and cohesion based on redesigned metrics of other researchers [29]. For coupling metrics, level of abstration of data dependency, events dependency, invokcation among methods in class, global dependency as well as inheritance dependency are used. Level of abstration of data dependency is defined as a sum of absolute values of DAF(c) and DAA(c) . DAF(c) is a list of fields for class what are abstract while DAA(c) is a list of attributes for method that are abstract. Events dependency is defined as a sum of absoulte values of MIE(c) and MIP(c), where MIE(c) is a list of remote methods called as expressions while MIP(c) is a list of remote methods called as real parameters. Invokcation among methods in class is defined as sum of absolute values of MP(c) and MR(c), where MP(c) is a list of formal parameters in abstract method and MR(c) represents abstract callback types list. Global dependency is defined the sum of absolute values of ARE(c) and ARP(c). In the metric, ARE(c) is used as expression in the form of a list external properties and ARP(c) is used as real parameters in the form of a list of external properties. The last metrics, inheritance dependency, is defined as the absolute value of IH(c) where IH(c) represents a list of parent classes. When it comes to metrics for cohesion, internal association ratio with extended methods to methods interaction is used. RCI is the sum of real interactions in the biggest rate of interactions inside class. Initially, the total number of actual interactions considered variable interactions and method to variables interactions only. The redefinition took methods to methods interactions into account. When it comes to the maximum possible interactions, it is defined as $MaxI(c) =^k C_2 +^a C_2 + (a * (b + c))$ where a represents the count of class field variables, b represents the count of method attributes variables, c represents the count of method formal parameters variables and k represents sum of functions in class c.

As the rapid development of Object Oriented technique, Chidamber and Kemerer designed six metrics to measure OOD from perspectives of class methods weighting, inheritance level in tree, total amount of children, dependency among classes, return value from class and inadequate association among methods. Meanwhile, metrics are evaluated using Weyukers metrics measurement principles. This metrics suite is desired to be used by professional software developers working on commercial projects [16].

For the purpose of narrowing gaps between static and dynamic metrics, Gunnalan et. al. [22] purposed the concept of pseudo dynamic metrics to predict dynamic actions in the early stage of software development lifecycle. There are three steps involving in calculating pseudo dynamic metrics. Step 1 is to obtain the static metrics for all the components by using automated tools. Step 2 is to analyse the operation profile for all the components. It can be estimated based on domain experts or programmers assessment. The last step is to multiply the static metrics values with the operational profile values.

A relatively new semantic metrics based on automatic analysis of natural language (NL) design specification for object-oriented system was presented by Gall et. al. [21] in 2008. Unlike past design metrics calculated from diagram format or during the implementation stage, it provides a preview of the software quality in the early stage of software development. A NL-based comprehension tool semMet is expended to calculate semantic metrics from design specifications. 11 metrics are defined in the paper along with descriptions on how to calculate them. Metrics covered in the paper are class domain complexity (CDC), relative class domain complexity (RCDC), semantic class domain entropy (SCDE), relative class domain entropy (RCDE), Logical relatedness of methods (LORM), Logical Disparity of members (LDM), Percentage of Shared Ideas (PSI), Percentage of Universal Ideas (PRI), Percentage of Universal Ideas(PUI), Percentage of Related Ideas(PCRC), Average proportion of Ideas Shared with Other Classes(APISOC).

From the perspective of users, good software is trustworthy. Tao and Chen introduced a metric model to measure system trustworthiness [48]. In their metric, they separated attributes related to trustworthiness into two groups, critical and non-critical attributes. Each group has different weight on influencing the overall trustworthy value. Attributes such as reliability, correctness, availability, controllability, security and so on are critical attributes. While attributes such as maintainability, portability etc. are non-critical attributes. Each attribute is in the interval of [1, 10] representing the degree of that attribute in a system. The metric uses minimum value in critical attributes as system trustworthiness.

2.8 Behavioural Profile

For the purpose of fetching key behavioural relationships in process models, Weidlich et. al. introduced weak order relations, strict order relations, exclusiveness relations as well as observation concurrency relations. Weak order relation restricts the sequence of nodes in process model. In addition to the weak order relation, if two notes are in a strict order relation, loop is not allowed. For an exclusiveness relation, a pair of nodes defined in this relation should not appear in the same trace. While for concurrency relation, nodes in a pair should appear concurrently. Strict order relations, exclusiveness relations and observation concurrency relations form the behavioural profile of process model [51].

Chapter 3

Research Approach

3.1 Factors contributing to reactivity

Since agents are situated in dynamic environments, it is crucial that they are able to respond to changes (that are relevant to the agents) in the environment in a timely fashion. When developing an agent system, reactivity can be maximized in several ways. The developer needs to make sure that the agent's plan library has plans to handle all external events from the environment(i.e. changes in the agent's beliefs due to perceived changes in the environment). Those plans have the external event as the trigger event. For instance, an agent-based weather alerting system (as in [37]) needs to deal with a range of events such as rain, wind, volcano ash, earthquake and so on. Assuming a sensor installed on an agent receives changes in the environment but the agent cannot make corresponding response to that change, the agent is not reactive to the environment in this case. It is a threat for an agent not being able to reactive to the environment. In some cases, not being reactive to the environment might be catastrophic. Robots used in fire rescue that do not have enough plans in a plan library may be destroyed by fire or falling items. For agents which involving in jobs requiring cooperation such as bomb detection, a failure in reactive to the environment of a single agent may destroy all other agents. Due to the variation and uncertainty of the environment, a designer or developer needs to ensure plans in plan library covers all states of the environment where an agent is suited.

For reactivity issue caused by lack of plans, it can be solved by increasing the number of plans for external events from the environment. However, reactivity during the goal-orientated behaviour is more complex as well as interesting. When dealing with a certain event, the agent should be designed in such a way that it commits to a certain courses of action as late as possible, i.e. to wait until the agent gets the most updated knowledge about the environment. In this case, we can ensure the agent performs most accurate actions. Let us demonstrate this with the following example. Assume that for a given event trigger te, I have designed two plans P1 and P2 which handle this event and are written as follows:

P1 $te : c \leftarrow a_1; a_2; a_3$ **Design Option 1**
P2 $te : \neg c \leftarrow a_1; a'_2; a'_3$

It is noted that in the first implementation option a_1, a_2, a_3, a'_2 and a'_3 are primitive actions.

This agent system can be developed in a different way to handle the same events by having subgoals in the plan body. These plans are now re-written as follows (the second design option):

P1 $te \leftarrow a_1; !sg$ **Design Option 2**
P2 $sg : c \leftarrow a_2; a_3$
P3 $sg : \neg c \leftarrow a'_2; a'_3$

Both design options would lead to the same ways of handling event te, i.e. by performing either $\langle a_1; a_2; a_3\rangle$ or $\langle a_1; a'_2; a'_3\rangle$. However, in the second design option (i.e. plans P1, P2 and P3 with subgoals) the agent does not commit to do either $\langle a_2; a_3\rangle$ or $\langle a'_2; a'_3\rangle$ until a_1 is completed. In contrast, with the first design option the agent makes this commitment earlier. This means in the case if there are any changes in the environment at the time after a_1 is completed (e.g. condition c no longer holds or vice versa), the agent fails to respond to this change (e.g. continues either doing $\langle a_2; a_3\rangle$ or $\langle a'_2; a'_3\rangle$). Therefore, the second design option makes the agent more reactive than the first one does. This example indicates that the number of subgoals in an agent plan library has an impact on how reactive the agent is at runtime.

The use of subgoals is also encouraged in agent design since it decouples a goal from its plan and makes it easy to add other plan choices later. However, just only turning primitive actions into subgoals does not merely improve the reactivity of an agent. Let us consider the following design.

P1 $te : c \leftarrow a_1; sg_2; a_3$ **Design Option 3**
P2 $te : \neg c \leftarrow a_1; sg'_2; a'_3$
P3 $sg_2 \leftarrow a_2$
P4 $sg'_2 \leftarrow a'_2$

In this example, although there are two subgoals in the agent's plans the actual behaviour of the agent in this example is identical to the one in the initial design. Therefore, a reactivity measure should only take subgoals with context conditions into account. A context condition also plays a role in contributing to the reactivity of an agent. For instance, if an agent has two plans $te : c_1 \leftarrow a_1$ and $te : c_2 \leftarrow a_2$, then its behaviour is still in some sense reactive, as which of a_1 and a_2 gets executed will depend on which of c_1 and c_2 are true in the current state of the environment.

3.2 Reactive Measures

3.2.1 Measure for Agents' Reactivities

We have previously discussed several factors that contribute to reactivities of an agent. In this section, A measure that can be applied to an agent is proposed, specifically an agent's plan library, to evaluate agent's reactivities. The proposed measure can be regarded as a static measure since it involves analysing source code, as opposed to dynamic measures which assess the characteristics of the software during execution [5].

The reactivity measure reflects whether an agent has plans to handle different events in the environment. Such events are considered as top triggering events in the library of agent's plans. Top triggering events are events which are not generated by any plan's body and they are often the external events that are significant to the agent (i.e. rain, wind, etc.) or an initial goal (i.e. go to university). As mentioned in section 3.1, plans for trigger events can increase reactivity while primary actions inside plans for triggering event can bring negative effect to reactivity. So we can have the reactivity measure on plans for the triggering event as follows:

$$R(_{TP}) = \frac{TP(te)}{A_{TP}(te)} \tag{3.1}$$

In the measure, $TP(te)$ is the number of plans for triggering event. Only plans containing actions or subgoals should be counted. $A_{TP}(te)$ is the number of primary actions without context condition in front of them. As can be seen in the measure, $R(_{TP})$ is direct proportional to TP(te), $R(_{TP}) \propto$ TP(te). Whereas $R(_{TP})$ is inverse proportional to $A_{TP}(te)$, $R(_{TP}) \propto 1/A_{TP}(te)$. If $A_{TP}(te)$ is 0, the measure turns out to be invalid. Therefore, a positive integer is added to the denominator to ensure that the denominator is not 0. This thesis picks 1 in this measure. However, any positive integer can be added to the denominator.

$$R(_{TP})' = \frac{TP(te)}{A_{TP}(te) + 1} \tag{3.2}$$

In equation 3.1, if $A_{TP}(te)_1 > 0$, $A_{TP}(te)_2 > 0$, $TP(te)_1 = TP(te)_2$ and $A_{TP}(te)_1 > A_{TP}(te)_2$, then $R(_{TP})_1 < R(_{TP})_2$. In equation 3.2, if $A_{TP}(te)_1 \geq 0$, $A_{TP}(te)_2 \geq 0$, $TP(te)_1 = TP(te)_2$ and $A_{TP}(te)_1 > A_{TP}(te)_2$, then $R(_{TP})'_1 < R(_{TP})'_2$. Under the condition that there does not exist primary action, a library which contains more plans for triggering events is more reactive. Therefore, adding a positive integer on denominator in the reactive measure does not affect the result when comparing two libraries.

If sub-goals or nested sub-goals exist in plans for triggering event, those sub-goal plans with context condition can increase agent reactivity. Actions in sub-goals without context condition in front of them can bring negative effect for agents in responding to the environment. The following measure can be used to calculate reactivity of sub-goals.

$$R(_{SGP}) = \frac{SGP(te)}{A_{SGP}(te)} \tag{3.3}$$

In the measure, $R(_{SGP})$ is the number of plans for sub-goals with context conditions after these sub-goals. $A_{SGP}(te)$ is the number of primary actions without context conditions in front of primary actions. As can be seen in the measure above, $R(_{SGP})$ is directly proportional to $SGP(te)$, $R(_{SGP}) \propto SGP(te)$. Whereas $R(_{SGP})$ is inverse proportional to $A_{SGP}(te)$, $R(_{SGP}) \propto A_{SGP}(te)$. If $A_{SGP}(te)$ is 0, the measure turns out to be invalid. Therefore, a positive integer is added to the denominator to ensure that the denominator is not 0. This thesis picks 1 in this measure. However, any positive integer can be added to the denominator.

$$R(_{SGP})' = \frac{SGP(te)}{A_{SGP}(te) + 1} \tag{3.4}$$

In equation 3.3, if $A_{SGP}(te)_1 > 0$, $A_{SGP}(te)_2 > 0$, $SGP(te)_1 = SGP(te)_2$ and $A_{SGP}(te)_1 > A_{SGP}(te)_2$, then $R(_{SGP})_1 < R(_{SGP})_2$. In equation 3.4, if $A_{SGP}(te)_1 \geq 0$, $A_{SGP}(te)_2 \geq 0$, $SGP(te)_1 = SGP(te)_2$ and $A_{SGP}(te)_1 > A_{SGP}(te)_2$, then $R(_{SGP})'_1 < R(_{SGP})'_2$. Under the condition that there does not exist actions without context condition in front of them, a library which contains more plans for subgoals is more reactive. Therefore, adding a positive integer on denominator in the reactive measure does not affect the result when comparing two libraries.

In calculating reactivity level for plans for single external event, we can sum up $R(_{TP})'$ and $R(_{SGP})'$. However, as plans for external event and sub-goals may have different weight on affecting reactivity. A constant variable k from 0 to 1 is provided as the weight of plans for external events. Thus, we can have the reactivity measure for triggering event as follows:

$$\begin{aligned} R(te) &= k \times R(_{TP})' + (1-k) \times R(_{SGP})' \\ &= k \times \frac{TP(te)}{A_{TP}(te) + 1} + (1-k) \times \frac{SGP(te)}{A_{SGP}(te) + 1} \end{aligned} \tag{3.5}$$

In practice, an agent may have different degrees of attention to different events in the environment, depending on requirements. For instance, if an earthquake event is crucial to an agent's operation, it is very important for the agent to have (many) plans to deal with such an event. On the other hand, the agent may not need to react to an event of raining. In order to capture this, we need to support differential weightings for different types of external events and allow the developer to specify them based on the system requirements. If there are no specific requirements regarding this aspect, all the external events can be assigned the same weight.

Due to the above reasons, one component of the reactivity metric involves counting the number of plans that handle the top triggering events. The other component involves counting the proportion of the number of subgoals in a plan against the number of primitive actions. This reflects the fact that an agent can also be considered to be reactive in the sense that it delays committing to a certain course of action as late as possible. A weight is bound to each component which allows the developer to specify whether one reactive component is more important than the other. The definition of reactivity measure for an agent is as follows:

Definition 1:

$$\begin{aligned} R(ag) &= \sum_{i=1}^{n} \alpha_i \times R(te)_i \qquad (3.6) \\ &= \sum_{i=1}^{n} \alpha_i \times (k \times R(_{TP_i})' + (1-k) \times R(_{SGP_i})') \\ &= \sum_{i=1}^{n} \alpha_i \times (k \times \frac{TP(te_i)}{A_{TP}(te_i)+1} + (1-k) \times \frac{SGP(te_i)}{A_{SGP}(te_i)+1}) \end{aligned}$$

where:

1. $R(ag)$: the reactivity measure of agent ag.
2. te_i: the i^{th} external event handled in the plan library of agent ag.
3. n: the total number of external events.
4. α_i: the weight for triggering event i^{th}, $\alpha_i \in [0,1]$.
5. k: the weight of reactivity to external events, $k \in [0,1]$.
6. $TP(te_i)$ number of alternative plans for triggering event.

7. $A_{TP}(te_i)$: total number of primitive actions without context condition in front of them in the triggering event plans to handle trigger event te_i.

8. $SGP(te_i)$: total number of plans for subgoals in the domain of triggering event te_i (excluding plans without context conditions).

9. $A_{SGP}(te_i)$: total number of actions without context condition in front of them under the subgoal domain of triggering event te_i.

As can be seen in the reactivity measure for agent, R(ag) is direct proportional to TP(te) and SGP(te) which can be expressed as R(ag) $\propto$ TP(te) and R(ag) $\propto$ SGP(te). R(ag) is reverse proportional to denumerator, $A_{TP}(te_i)$ and $A_{SGP}(te_i)$, which can be expressed as: R(ag) $\propto$ 1 / $A_{TP}(te_i)$ and R(ag) $\propto$ 1 / $A_{SGP}(te_i)$. Due to the fact that reactivity can be increased constantly by adding plans for new triggering event(increase n), the range of R(ag) is in the range of $[0, +\infty)$.

3.2.2 Reactivity Measure for Goal-oriented Behaviour

As mentioned above, if the number of triggering events is taken into consideration, there does not exist a maximum value for agent reactivity. However, if we can guarantee that all events happened in the environment can be handled by an agent, it is possible to know how reactive the agent is by comparing its reactivity with the maximum reactivity the agent can reach in the scope of goal-oriented behaviour. The benefit of reactivity measure for goal-oriented behaviour is we can know the reactivity of an agent without comparing it with other agents' plan library.

Upon receiving a triggering event, agent starts to perform goal-oriented behaviour. The goal of the agent is to make proper response to received event. During goal-oriented behaviours, if all actions can be performed immediately after context condition checking, that means the agent gets the latest status of the environment. In this case, the agent performs actions as supposed. As can be seen in section 3.2.1, reactivity for an agent can be measured by :

$$\begin{aligned} R(ag) &= \sum_{i=1}^{n} \alpha_i \times R(te)_i && (3.7)\\ &= \sum_{i=1}^{n} \alpha_i \times (k \times R(_{TP_i})' + (1-k) \times R(_{SGP_i})') \\ &= \sum_{i=1}^{n} \alpha_i \times (k \times \frac{TP(te_i)}{A_{TP}(te_i)+1} + (1-k) \times \frac{SGP(te_i)}{A_{SGP}(te_i)+1}) \end{aligned}$$

Assume we have a plan library for the same agent that can handle same number of external event(identical n, α_i and k), we can have a measure for maximum reactivity as follows:

$$\begin{aligned} R_{max}(ag) &= \sum_{i=1}^{n} \alpha_i \times R_{max}(te)_i && (3.8)\\ &= \sum_{i=1}^{n} \alpha_i \times (k \times R_{max}(_{TP_i})' + (1-k) \times R_{max}(_{SGP_i})') \\ &= \sum_{i=1}^{n} \alpha_i \times (k \times \frac{TP_{max}(te_i)}{A_{TP_{max}}(te_i)+1} + (1-k) \times \frac{SGP_{max}(te_i)}{A_{SGP_{max}}(te_i)+1}) \end{aligned}$$

We can thus create a reactivity measure for goal-oriented behaviour by dividing current agent's reactivity by the maximum reactivity it can reach:

$$R_{gob}(ag) = \frac{R(ag)}{R_{max}(ag)} \tag{3.9}$$

When agent is not reactive at all, $R_{gob}(ag) = 0$. However, if agent is most reactive, then $R_{gob}(ag) = 1$. A figure below shows the distribution of the measure.

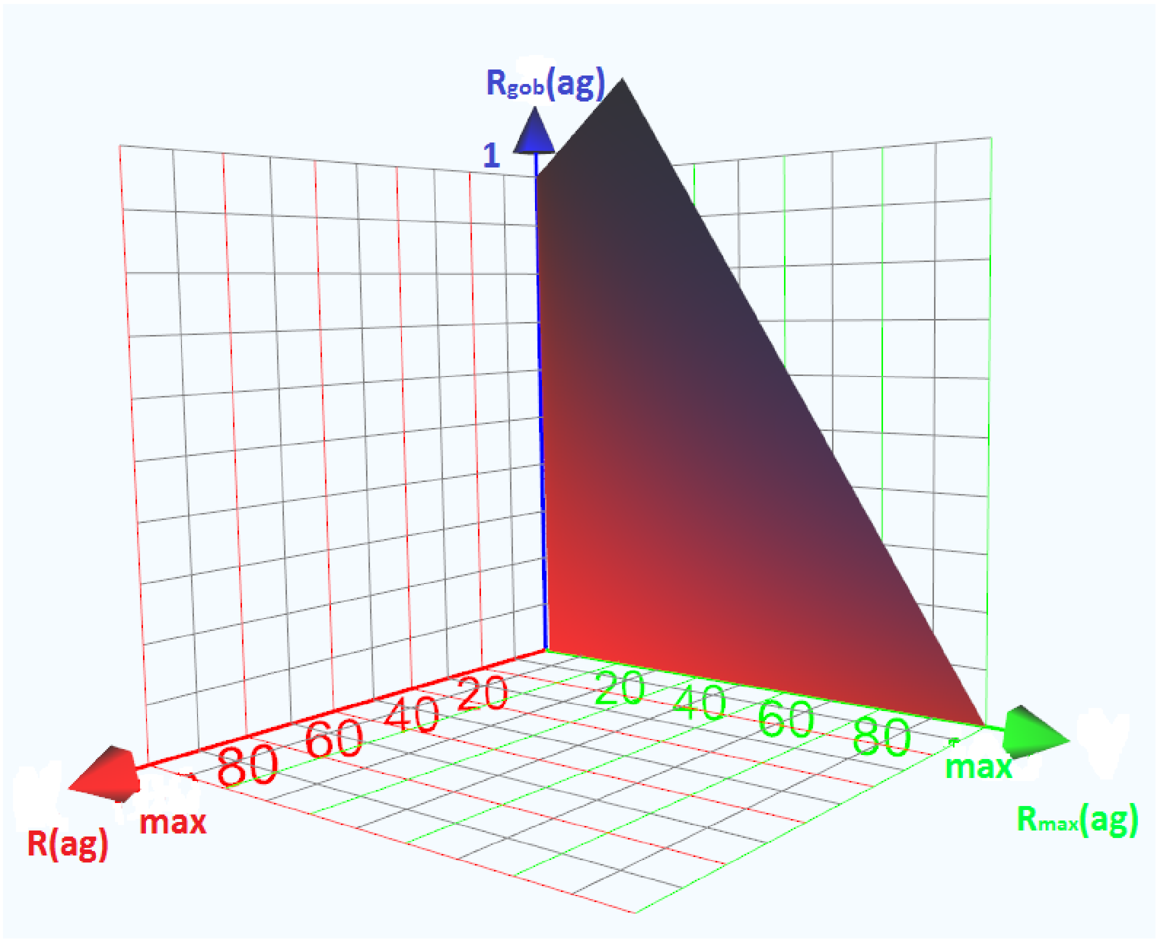


Figure 3.1: Goal-oriented Behaviour Reactivity Measure Distribution

3.2.3 Domain Graph for Triggering Event

Measures on agent reactivities are based on domains of triggering events and subgoals. The definition of triggering events and subgoals domains stems from the hierarchical structure of BDI agents' plan: events are handled by plans which generate further events handled by other plans and so on. The domain of a trigging event is represented as a graph that contains sub-goal nodes, context condition nodes and action nodes that are reachable from the trigging event. Figure 3.2 shows an example of a domain triggering events and subgoals for the following plans.

P1 $TE : C1 \leftarrow SG1$
P2 $TE : \neg C1 \leftarrow A1$
P3 $SG1 : C2 \leftarrow A2; A3; SG2$
P4 $SG2 : C3 \leftarrow A4$
P5 $SG2 : \neg C3 \leftarrow SG1$

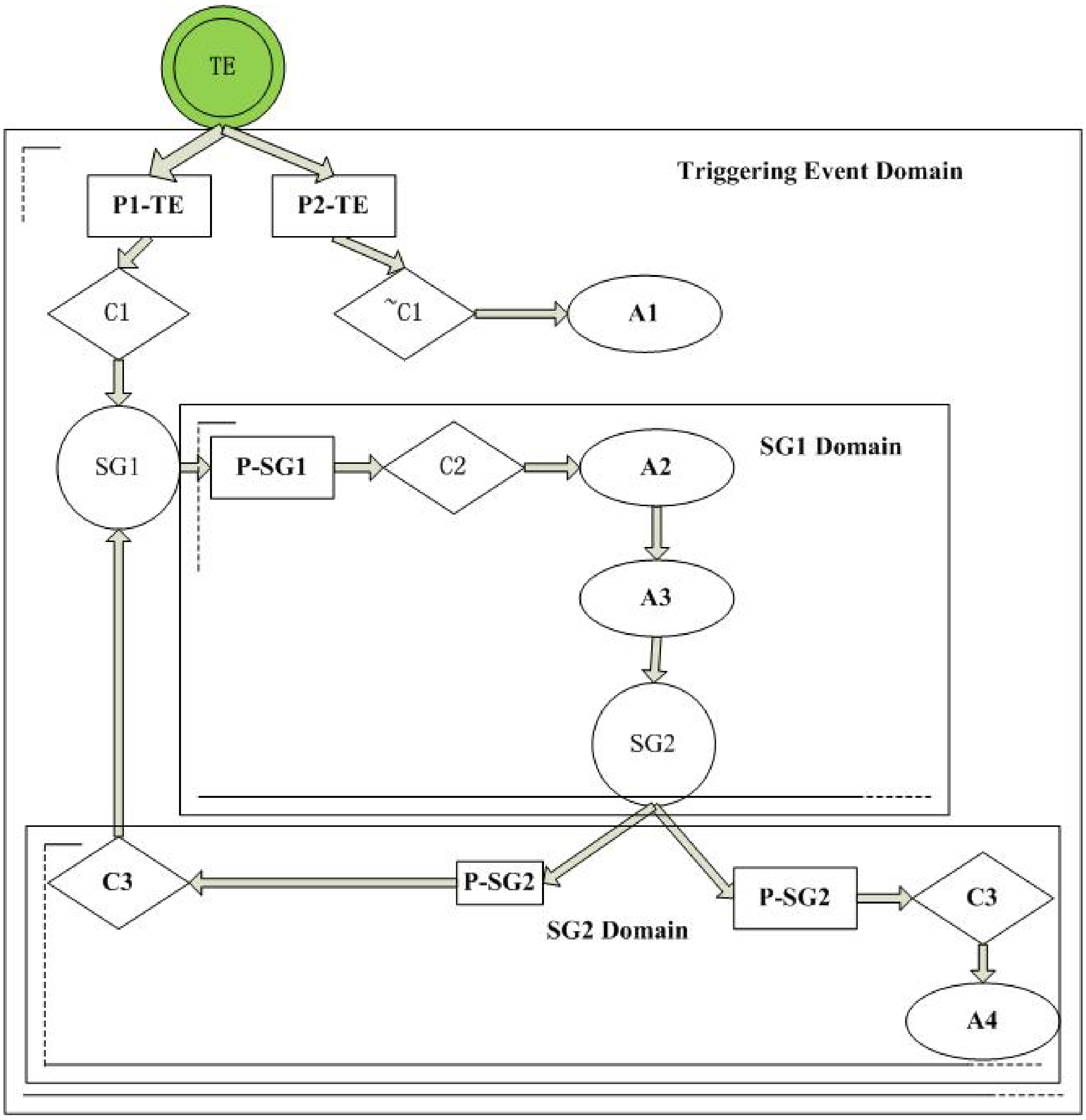


Figure 3.2: An example of a domain of triggering events and subgoals

Based on the measure, the reactivity of an agent which has the above plans P1–P5 is 0.88. The reactivity values of each design options discussed in section 3.1 are 0.14 (design options 1 and 3) and 0.45 (design option 2). These justify the fact that having subgoals (that are handled by different plans in different situations) increase the reactivity of an agent.

As can be seen node types can be a trigging event (TE), plan (e.g. P-SG1), context conditions (C), sub-goals(SG) and actions (A). The definition of the domain of a subgoal is given as follows:

Definition 2:

The domain of a subgoal is defined as all the subgoals, context conditions and actions in the plans handling the subgoal that are reachable from the sub-goal. If there are nested sub-goals, then nested subgoals, context conditions and actions belong to the nearest sub-goal domain it contains.

For design option 1 mentioned in section 3.1, we can have the domain graph for triggering event in figure 3.3. In order to create the domain graph, we fetch triggering event or goal, TE, from the design option 1 at first. As TE has two available plans depending on different returned results of context condition C, two branches, P1-TE and P2-TE, with context condition C are linked to TE. As plan P1 contains primary actions a_1, a_2 and a_3, these actions can be joined to context condition C under P1-TE. As P2 contains primary actions a_1, a'_2 and a'_3, these actions can be joined to context condition $\neg C$ under P2-TE. As there does not exist sub-graph in design option 1, we can say all actions, plans and context conditions are in the domain of TE.

In figure 3.3, if TE is received by the agent, plan P1-TE and P2-TE are available for it. By checking the context condition C, an agent can either perform $< a_1, a_2, a_3 >$ or $< a_1, a'_2, a'_3 >$. Therefore, the logic in domain graph is consistent with design option 1.

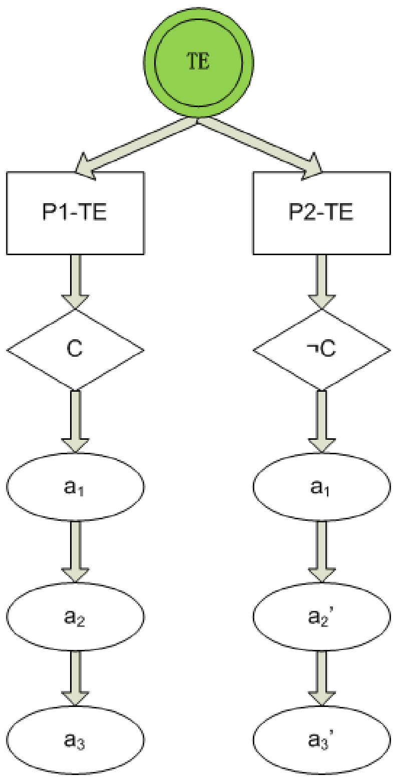


Figure 3.3: Domain Graph for Design Option 1

For design option 2 mentioned in section 3.1, we can have the domain graph for triggering event in figure 3.4. In order to create the domain graph, we fetch triggering event or goal, TE, from the design option 2 at first. Unlike design option 1, in design option 2, TE has one plan, P-TE, available only. As there are no context conditions before a_1, a_1 can be performed once TE is received by agent. So we can join P-TE to TE, then joining a_1 to plan P-TE. As a subgoal SG follows after a_1, a sub-goal SG can be created after a_1. After creating the sub-goal node, we can focus on plans for the subgoal SG. We noticed a context condition C in subgoal SG. Thus, we can create two branches for boolean value of returned from context condition C. For C is true, a_2, a_3 can be performed. So actions a_2 and a_3 can be joined to C. For C is false, a_2' and a_3' can be performed. So actions a_2 and a_3 can be joined to $\neg C$. As design option 2 has a subgoal SG, sub-domain SG contains P-SG, C, $\neg C$, a_2, a_3, a_2' and a_3'. What worth mentioning is all subgoals, actions, plans and context conditions still belong to domain TE.

Upon tracing the flow of domain graph 3.4 for TE, a_1 can be executed as long as TE is received. When context condition C is true, a_2 and a_3 can be performed. Otherwise, a_2' and a_3' can be performed. By comparing logics with design option 2, logics in domain graph is consistency with design option 2.

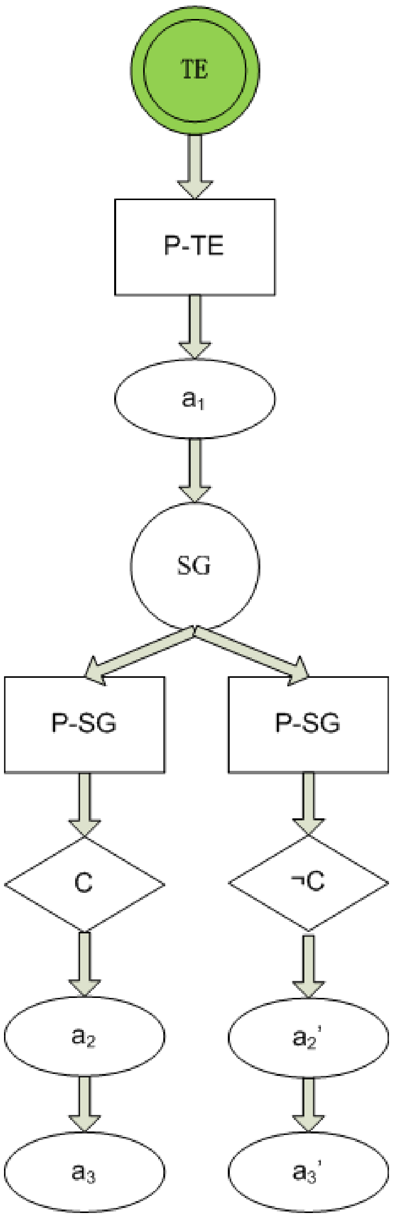


Figure 3.4: Domain Graph for Design Option 2

Let us focus on the domain graph 3.5 for design option 3 in section 3.1. As we can see, two plans, P1-TE and P2-TE, are available for triggering event TE. Hence we can create two branches after TE with context condition C and $\neg C$. Action a_1 can be performed in both plan, so we can add a node 'a_1' after C and another node 'a_1' after $\neg C$. As two distinct sub-goals, SG_2 and SG_2', follow after a_1 in P1-TE and a_1 in P2-TE respectively, two sub-domains can be created for them. Because there does not exist context condition in plan for SG_2 and plan for SG_2', a_2 and a_3 can be linked to $P - SG_2$ directly. At the same time, a_2' and a_3' can be linked to $P - SG_3$ directly. By looking at the figure, there are two sub-domains, one for sub-goal SG_2 containing $P - SG_2$, a_2 and a_3 and another for sub-goal SG_2' containing $P - SG_2'$, a_2' and a_3'. All subgoals, actions, plans and context conditions are in the domain TE as well.

In order to provide the correct mapping between design option 3 and domain graph 3.5, we trace the flow of the domain graph. When TE is received by agent, there are two options available for it. When C is true, agent performs action a_1, a_2 and a_3. Otherwise, agent performs a_1, a_2' and a_3'. By comparing the flow mentioned in design option 3, we can say the domain graph keeps functionality in design option 3.

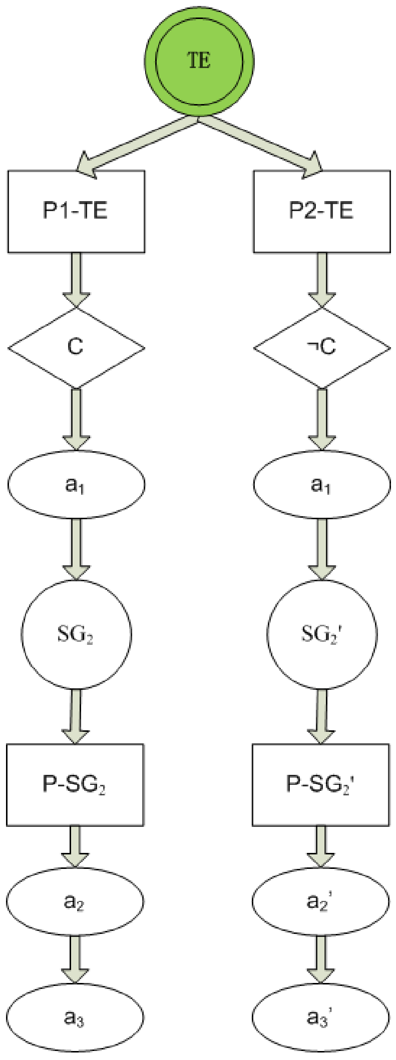
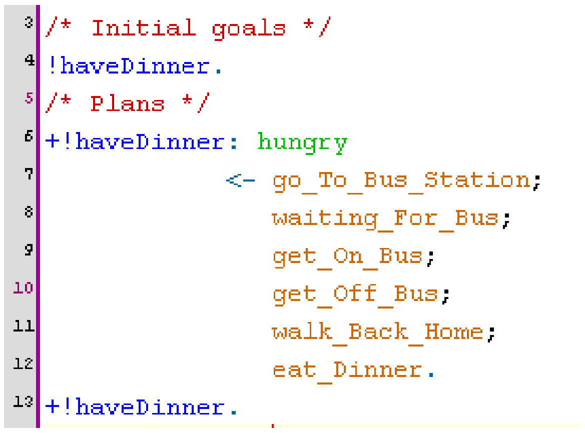


Figure 3.5: Domain Graph for Design Option 3

3.2.4 Example on reactivity calculation

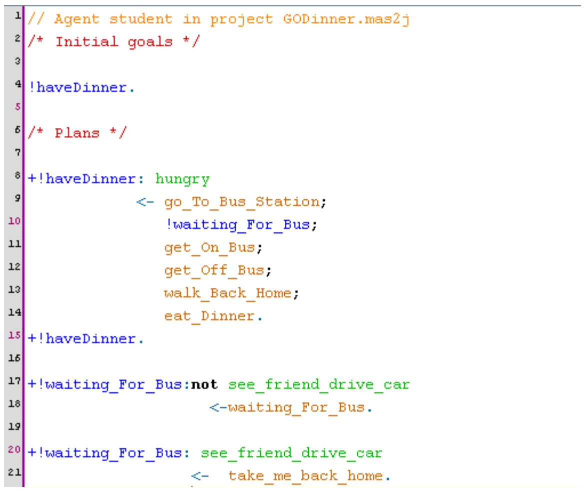
Three plan libraries with different implementation style on student going for dinner are used to demonstrate the calculation using the measure for reactivity.



```
/* Initial goals */
!haveDinner.
/* Plans */
+!haveDinner: hungry
            <- go_To_Bus_Station;
               waiting_For_Bus;
               get_On_Bus;
               get_Off_Bus;
               walk_Back_Home;
               eat_Dinner.
+!haveDinner.
```

Figure 3.6: Plan library of student going back for dinner(without subgoal)

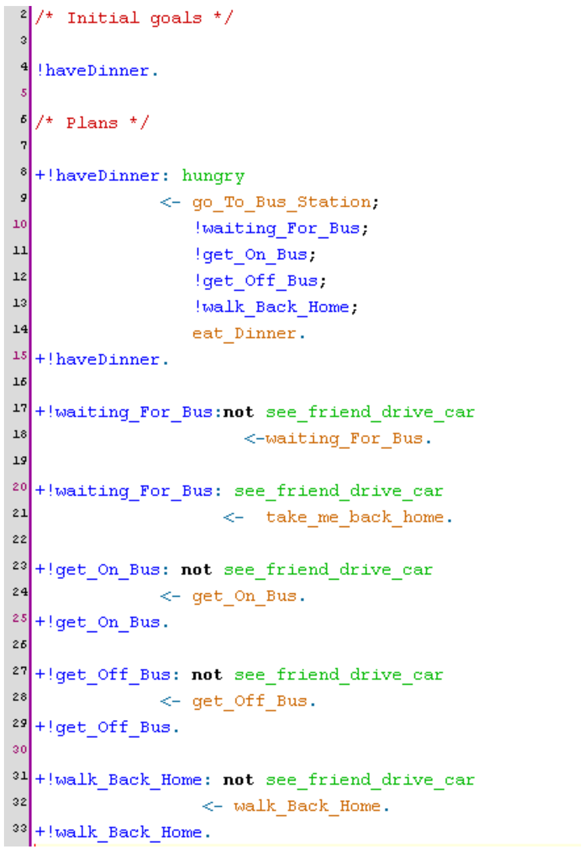
In plan library 3.6, there is not sub-goals, and only one valid alternative plan for initial goal (with the context condition and has action after it). There are six actions defined on the domain the top trigging event. So the reactivity of the agent is: $R(student) = 0.5 \times \frac{1}{6+1} + 0.5 \times \frac{0}{1} = 0.071$



```
// Agent student in project GODinner.mas2j
/* Initial goals */

!haveDinner.

/* Plans */

+!haveDinner: hungry
            <- go_To_Bus_Station;
               !waiting_For_Bus;
               get_On_Bus;
               get_Off_Bus;
               walk_Back_Home;
               eat_Dinner.
+!haveDinner.

+!waiting_For_Bus:not see_friend_drive_car
                   <-waiting_For_Bus.

+!waiting_For_Bus: see_friend_drive_car
                 <-  take_me_back_home.
```

Figure 3.7: Plan library of student going back for dinner(with one subgoal)

In plan library 3.7, action of waiting for bus is placed in a sub-goal with two alternative plans. This makes it possible for agent to check whether a perception of see_friend_drive_car is received or not. The plan library contains one sub-goal, one alternative plan for initial goal, 5 actions in initial goal, two valid alternative plans and two actions in sub-goal alternative plans. So the reactivity of it is: $R(student) = 0.5 \times \frac{1}{5+1} + 0.5 \times \frac{2}{2+1} = 0.416$



```
/* Initial goals */

!haveDinner.

/* Plans */

+!haveDinner: hungry
            <- go_To_Bus_Station;
               !waiting_For_Bus;
               !get_On_Bus;
               !get_Off_Bus;
               !walk_Back_Home;
               eat_Dinner.
+!haveDinner.

+!waiting_For_Bus:not see_friend_drive_car
                    <-waiting_For_Bus.

+!waiting_For_Bus: see_friend_drive_car
                  <-  take_me_back_home.

+!get_On_Bus: not see_friend_drive_car
            <- get_On_Bus.
+!get_On_Bus.

+!get_Off_Bus: not see_friend_drive_car
            <- get_Off_Bus.
+!get_Off_Bus.

+!walk_Back_Home: not see_friend_drive_car
                <- walk_Back_Home.
+!walk_Back_Home.
```

Figure 3.8: Plan library of student going back for dinner(with four subgoals)

In plan library 3.8, actions of 'waiting for bus', 'get on bus', 'get off bus' and 'walk back home' are all moved to sub-goals. This makes it possible for agent to check whether a perception of see_friend_drive_car is received or not. If the agent receives this perception, actions such as get on bus, get off bus and walk back home cannot be performed. This plan consists 5 alternative plans for sub-goals, 1 valid alternative plan for initial goal , two actions under initial goal and 5 actions in subgoals. So the reactiveness of it is: $R(student) = 0.5 \times \frac{1}{2+1} + 0.5 \times \frac{5}{5+1} = 0.576$

3.2.5 The reactivity algorithm

Based on the reactivity measure described in the previous section, we propose an algorithm to calculate the reactivity of an agent's plan library as follows. For each top triggering event in the plan library, we need to construct a domain graph for it (as in figure 3.2 in the previous section).

```
    function ReactivityCalculation(PlanLibrary)
1       if PlanLibrary is empty then
2           return 0
3       end if
4       set sum to 0
5       set alpha to 1 //default to 1
6       for each TEgraph in PlanLibrary
7           set sumTeAlt to 0
8           set sumSubAlt to 0
9           set sumTEAction to 1
10          set sumSubAction to 1
11          if TEgraph has no node inside of it then
12              continue
13          end if
14          DFSCalcuation(TEgraph, triggeringEventNodeId,
                                  triggerEventNodeType)
15          if not default TriggingEvent Weight is used then
16              set alpha to User Input
17          end if
18          set sum to sum + alpha × (k × $\frac{sumTeAlt}{sumTeAction}$ + (1-k) × $\frac{sumSubAlt}{sumSubAction}$)
19      end for
20      return sum
    end function
```

Figure 3.9: Calculating the reactivity of an agent's plan library

The algorithm given in figure 3.9 compute the reactivity of an agent's plan library based on the reactivity measure. It first checks whether the plan library is empty or not (line 1). If the agent's plan library is empty, the reactivity of agent should be 0 (line 2). The *sum* variable initialized to 0 (line 4) is used to store the reactivity result. Alpha is set to 1 by default to represent all triggering events having the same effect on the agent's reactivity (line 5). The for loop in the algorithm (lines 6-19) calculates the reactivity of each trigging event graph in the plan library and add to sum (line 18). In the loop, it resets global variables *sumTeAlt*, *sumSubAlt*, *sumTEAction* and *sumSubAction* at the beginning of each iteration (lines 7- 10). If a triggering event's domain graph *TEgraph* does not contain any element in it (line 11), this triggering event's reactivity is 0. Under this condition, the algorithm stops (line 12) and starts to calculate the reactivity for the rest of the *TEgraph*. If *TEgraph* has some elements, a depth first search (line 14) is called on the *TEgraph* to set the value of *sumTeAlt*, *sumSubAlt*, *sumTEAction* and *sumSubAction*. The *TriggeringEventNodeId* represents the unique identifier of the node representing the triggering event in *TEgraph* and the *TriggerEventNodeType* represents the node's type which is Trigger Event (line 14). What worth mentioning is that the algorithm also allow users to specify each triggering event's weight (lines 15-17).

```
Procedure: DFSCalcuation(TEgraph, currentNodeId, domainNodeType)
  if number of visitedNodes in TEgraph is not equal to number of nodes in TEgraph then
    set CurrentNode TO the Node with currentNodeId
    if CurrentNode is Not visited And Exist then
      if domainNodeType is SUBGOAL AND CurrentNode is ContextCondition
        AND CurrentNode→NEXT is Action or Sub-Goal then
        increment sumSubAlt by 1
      else if domainNodeType is TRIGGERINGEVENT AND CurrentNode is
            PLAN type then increment sumTEAlt by 1
      else if CurrentNode Type is ACTION then
        if domainNodeType is SUBGOAL then increment sumSubAction by 1
        else if domainNodeType is TRIGGERINGEVENT then
          increment sumTEAction by 1
        end if
      end if
      set CurrentNode to VISITED
      if CurrentNode Type is SUBGOAL OR TRIGGERINGEVENT then
        for each Edge IN CurrentNode's Outgoing Edge do
          if Edge Type is ALTER_PLAN_EDGE AND destinationNode is
            NOT visited then
            DFSCalcuation(TEGraph, Edge's destination Node Id, CurrentNode Type)
          end if
        end for
      end if
      for each Edge IN CurrentNode's OutgoingEdge do
        if Edge Type is SEQUENCE_EDGE AND destination Node is Not visited then
          DFSCalcuation(TEGraph, Edge's destination Node Id, domainNodeType)
        end if
      end for
    end if
  end if
end procedure
```

Figure 3.10: Performing a depth first search on the domain graph to compute the reactivity

The algorithm given in figure 3.10 traverses the *TEgraph* using depth first search combined with alternative plan priority algorithm. In the algorithm, if not all the nodes in *TEgraph* are visited (line 1), then it starts to execute. The current node is fetched from the *TEGraph* according to the current node identifier (line 2). If the current node does exist but is not being visited (line 3), sumTeAlt, sumSubAlt, sumTEAction or sumSubAction will be incremented based on current node's type. If the current node is in the domain of a sub-goal, it is a context condition node and its next node is an action or sub-goal node (line 4), then there exists a valid sub-goal alternative plan (increment sumSubAlt by 1). If the current node is in the domain of the trigging event and it is a plan node, then it is a valid triggering event alternative plan (increment sumTEAlt by 1). If the current node is an action node and is in the domain of sub-goal, then it is a sub-goal action node (lines 8-10). On the other hand, if the current action node is in the domain of triggering event, then it belongs to the triggering event (lines 11-12). The current node is set to "visited" (line 14) before we go to search for its child nodes. If the current node is a sub-goal node or a trigging event node, alternative plan paths are searched based on the depth first search in priority (lines 15-21). In addition, the domain type is set to current node type (line 18). After searching current the sub-goal or triggering event node's alternative plans, other nodes that are directly link to current node (any type of current node) will become a start node and have DFS algorithm implemented on them (lines 22-26).

3.3 Agent Behavioural Preservation

An agent plan library provides possible plans for handling goals that agent can achieve. However, in the process of enhancing agent reactivity, agent may lose the ability of achieving original goals. For instance, a student can go back home from university by walking to bus stop, getting on bus, waiting for bus to destination, getting off the bus and walking back home.

```
+!go_back_home : at_university
               ← walk_to_bus_stop;
                 get_on_bus;
```

```
                       stay_on_bus;
                       get_off_bus;
                       walk_back_home.
+!go_back_home : near_home
                    ← turn_around;
                       walk_back_home.
```

However, if actions get_on_bus and wait_for_bus_to_destination are swapped after reactivity of the plan being enhanced, the student can no longer perform actions in the sequence as before.

```
+!go_back_home : at_university
                    ← walk_to_bus_stop;
                       !try_to_stay_on_bus;
                       !get_on_bus;
                       get_off_bus;
                       walk_back_home.
+!get_on_bus : bus_arrived
                 ← get_on_bus.
+!try_to_stay_on_bus: on_bus
                       ← stay_on_bus.
+!go_back_home : near_home
                    ← turn_around;
                       walk_back_home.
```

In the enhanced plan library, student can perform actions <walk_to_bus_stop, stay_on_bus, get_on_bus, get_off_bus, walk_back_home>, <walk_to_bus_stop, get_on_bus, get_off_bus, walk_back_home>, <walk_to_bus_stop, stay_on_bus, get_off_bus, walk_back_home> , <walk_to_bus_stop, get_off_bus, walk_back_home> or <turn_around, walk_back_home >. However, it is impossible to perform actions <walk_to_bus_stop, get_on_bus, stay_on_bus, get_off_bus, walk_back_home>, which could be performed if original plan library was used.

Therefore, a definition is required to preserve agent's orginal behaviour while peforming reactivity enhancement. Inspired by business profiles on process models [50], I am going to propose the definition of agent behavioural profile which can be used for fetching relationships of actions in agent plan library. This thesis, we are interested in how behaviours can be reserved when performing reactivity enhancement. According to my research, loop in agent system is not a factor that can affect system reactivity. Therefore, we assume there are no loops in agent plan library.

3.3.1 Behavioural Profile for Agent Plan Library

Definition 1 (Weak Order Relation). Given an agent plan library P, the weak order relation $\succ_P$ is defined as $\subseteq$ A $\times$ A $\times$ C, where A is a set of all actions and C is a set of all context conditions in agent plan library. $<$ x, y, c $>$ *in* $\succ_P$ if and only if there exists a plan in P where c is the context condition and action x precedes action y in the plan. x $\overset{c}{\succ}_P$ y can be used as a shorthand for $<$ x, y, c $>$ in weak order relation.

In weak order relation, actions in tuples can be related to each other directly or indirectly. For instance, actions in weak order relation can have no other actions between them or have one or more actions between them. Weak order relation restricts the sequence of actions as well as conditions of what actions can be performed.

Definition 2 (Strict Order Relation). Given an agent plan library P, the strict order relation $\rightsquigarrow_P$ is defined as $\subseteq$ A $\times$ A $\times$ C, where $<$ x, y, c $>$ *in* $\rightsquigarrow_P$ if and only if there exists a plan in P where x $\overset{c}{\succ}_P$ y and y $\overset{c}{\not\succ}_P$ x. x $\overset{c}{\rightsquigarrow}_P$ y can be used as a shorthand for <x, y, c> in strict order relation.

The same as weak order relation, strict order relation enforces the sequence of actions. In addition, if two actions are defined in strict order relation, loop is not allowed in these two actions. We consider a pair is reverse strict order, denoted by x $\overset{c}{\rightsquigarrow}_P^{-1}$ y, if and only if y $\overset{c}{\rightsquigarrow}_P$ x.

Definition 3 (Exclusiveness Relation). Given an agent plan library P, the exclusiveness relation $+_P$ is defined as $\subseteq$ A $\times$ C $\times$ A $\times$ C, where <x, c, y, c'> *in* $+_P$ if and only if there exists a plan in P where x $\overset{c}{\not\succ}_P$ y and y $\overset{c'}{\not\succ}_P$ x. x $\underset{c'}{\overset{c}{+}}_P$

y can be used as a shorthand for <x, c, y, c'> in exclusiveness relation.

According to the definition of exclusiveness relation, actions in exclusiveness relation should not appear in weak order relation, strict order relation or concurrency relation. Exclusiveness relation appears after a context condition. Based on the result of context condition, agent can choose a most suitable plan among multiple plans and perform actions under that plan.Thus, actions in alternative plans will not be performed by agent.

Definition 4 (Concurrency Relation). Given an agent plan library P, the concurrency relation $\|_P$ is defined as $\subseteq$ A $\times$ A $\times$ C, where <x, y, c> *in* $\|_P$ if and only if there exists a plan in P where x $\overset{c}{\succ}_P$ y and y $\overset{c}{\succ}_P$ x. x $\overset{c}{\|}_P$ y can be used as a shorthand for <x, y, c> in concurrency order relation.

Based on the definition above, x can happen before y or y can happen before x under the condition of c. That means, if a pair of actions is in concurrency relation, there is no occuring sequence of actions. For a single action, it can either be defined as exclusive to itself or concurrent to itself, i.e., $(a \underset{c}{\overset{c}{+}}_P a)$ or $(a \overset{c}{\|}_P a)$. If an action is exclusive to itself, this action can not be repeated under specific context condition. On the other hand, if an action is concurrent to itself, this action can be repeated under certain condition.

Strict order relation, exclusiveness relation and concurrency relation make up the behavioural profile for agent plan library. In order words, behavioural profile $\mathbb{B}_P$ is a set of $\{\rightsquigarrow_P, +_P, \|_P\}$.

Definition 5 (Relationship Transitivity) Given an agent plan library P, for all $\{x, y, z\} \subseteq$ A, if x $\overset{c}{\succ}_P$ y and y $\overset{c}{\succ}_P$ z, then x $\overset{c}{\succ}_P$ z. For strict order relation, if x $\overset{c}{\rightsquigarrow}_P$ y and y $\overset{c}{\rightsquigarrow}_P$ z, then x $\overset{c}{\rightsquigarrow}_P$ z. For concurrency relation, if x $\overset{c}{\|}_P$ y and y $\overset{c}{\|}_P$ z, then x $\overset{c}{\|}_P$ z.

According to the definition of relationship transitivity, relationship transitivity applies to weak order relation, strict order relation and concurrency relation, but not for exclusiveness relation.

Definition 6 (Relationship Reversibility) Given an agent plan library P, for all $\{x, y\} \subseteq$ A and $\{c, c'\} \subseteq$ C, if x $\underset{c'}{\overset{c}{+}}_P$ y, then y $\underset{c}{\overset{c'}{+}}_P$ x, vice versa. For concurrency relation, if x $\overset{c}{\|}_P$ y, then y $\overset{c}{\|}_P$ x, vice versa.

Definition of relationship reversibility means the order of actions in exlusiveness and concurrency relation does not affect behavioural profile.

3.3.2 Examples for Creating Behavioural Profile

As true concurrency does not exist in Jason, we will not cover concurrency relationship in examples.

3.3.2.1 Example One

Example of student going back home in section 3.3 can be used to demonstrate the process of constructing agent behavioural profile. According to the plan library, we can create agent behavioural profile as follows:

Strict Order relation: $\{walk\_to\_bus\_stop \overset{at\_university}{\rightsquigarrow_P} get\_on\_bus, get\_on\_bus \overset{at\_university}{\rightsquigarrow_P} stay\_on\_bus, stay\_on\_bus \overset{at\_university}{\rightsquigarrow_P} get\_off\_bus, get\_off\_bus \overset{at\_university}{\rightsquigarrow_P} walk\_back\_home, turn\_around \overset{near\_home}{\rightsquigarrow_P} walk\_back\_home\}$

Exclusiveness relation: $\{walk\_to\_bus\_stop \overset{at\_university}{\underset{near\_home}{+}}{}_P turn\_around, walk\_to\_bus\_stop \overset{at\_university}{\underset{near\_home}{+}}{}_P walk\_back\_home, get\_on\_bus \overset{at\_university}{\underset{near\_home}{+}}{}_P turn\_around, get\_on\_bus \overset{at\_university}{\underset{near\_home}{+}}{}_P walk\_back\_home, stay\_on\_bus \overset{at\_university}{\underset{near\_home}{+}}{}_P turn\_around, stay\_on\_bus \overset{at\_university}{\underset{near\_home}{+}}{}_P walk\_back\_home, get\_off\_bus \overset{at\_university}{\underset{near\_home}{+}}{}_P turn\_around, get\_off\_bus \overset{at\_university}{\underset{near\_home}{+}}{}_P walk\_back\_home\}$

Concurrency relation: $\{\phi\}$

3.3.2.2 Example Two

For the purpose of comparision, the enhanced agent plan library for student going back home in section 3.3 is used. Agent behavioural profile for the enhanced agent plan library can be created as follows:

Strict Order relation: $\{walk\_to\_bus\_stop \overset{at\_university\&on\_bus}{\rightsquigarrow_P} stay\_on\_bus, stay\_on\_bus \overset{at\_university\&on\_bus\&bus\_arrived}{\rightsquigarrow_P} get\_on\_bus, get\_on\_bus \overset{at\_university\&bus\_arrived}{\rightsquigarrow_P} get\_off\_bus, get\_off\_bus \overset{at\_university}{\rightsquigarrow_P} walk\_back\_home, turn\_around \overset{near\_home}{\rightsquigarrow_P} walk\_back\_home\}$

Exclusiveness relation: $\{walk\_to\_bus\_stop \overset{at\_university}{\underset{near\_home}{+}}{}_P turn\_around, walk\_to\_$

$bus\_stop \overset{at\_university}{\underset{near\_home}{+}}{}_P walk\_back\_home, get\_on\_bus \overset{at\_university\&bus\_arrived}{\underset{near\_home}{+}}{}_P turn\_around,$
$get\_on\_bus \overset{at\_university\&bus\_arrived}{\underset{near\_home}{+}}{}_P\ walk\_back\_home, stay\_on\_bus \overset{at\_university\&on\_bus}{\underset{near\_home}{+}}{}_P$
$turn\_around, stay\_on\_bus \overset{at\_university\&on\_bus}{\underset{near\_home}{+}}{}_P walk\_back\_home, get\_off\_bus \overset{at\_university}{\underset{near\_home}{+}}{}_P$
$turn\_around, get\_off\_bus \overset{at\_university}{\underset{near\_home}{+}}{}_P\ walk\_$
$back\_home\}$

Concurrency relation: $\{\phi\}$

3.3.2.3 Comparison

As can be seen in the set of strict order relation in first example, *get_on_bus* has strict order relation with *stay_on_bus* ($get\_on\_bus \overset{at\_university}{\rightsquigarrow_P} stay\_on\_bus$). However, in the second example, relationship between *get_on_bus* and *stay_on_bus* is defined as $stay\_on\_bus \overset{at\_university\&on\_bus\&bus\_arrived}{\rightsquigarrow_P} get\_on\_bus$, which conflicts which the strict order relation defined in the first example. Therefore, agent behavioural in first agent plan library is not preserved in the second agent plan library despite the fact that agent reactivity has been enhanced.

3.3.2.4 Resolve Conflict

For the purpose of resolving the conflict in agent plan libraries, we can simply swap subgoals !*get_on_bus* and !*try_to_stay_on_bus* in the enhanced agent plan library.

```
+!go_back_home : at_university
               ← walk_to_bus_stop;
                 !get_on_bus;
                 !try_to_stay_on_bus;
                 get_off_bus;
                 walk_back_home.
+!get_on_bus : bus_arrived
             ← get_on_bus.
+!try_to_stay_on_bus: on_bus
                 ← stay_on_bus.
```

```
11 +!go_back_home : near_home
12                ← turn_around;
13                  walk_back_home.
```

Now, let's create the behavioural profile for the fixed agent plan library:
Strict Order relation: $\{\mathbf{walk\_to\_bus\_stop} \overset{\mathbf{at\_university\&bus\_arrived}}{\rightsquigarrow}_{\mathbf{P}} \mathbf{get\_on\_bus}, \mathbf{get\_on\_bus} \overset{\mathbf{at\_university\&bus\_arrived\&on\_bus}}{\rightsquigarrow}_{\mathbf{P}} \mathbf{stay\_on\_bus}, \mathbf{stay\_on\_bus} \overset{\mathbf{at\_university\&on\_bus}}{\rightsquigarrow}_{\mathbf{P}} \mathbf{get\_off\_bus}, get\_off\_bus \overset{at\_university}{\rightsquigarrow}_P walk\_back\_home, turn\_around \overset{near\_home}{\rightsquigarrow}_P walk\_back\_home\}$
Exclusiveness relation:$\{walk\_to\_bus\_stop \overset{at\_university}{\underset{near\_home}{+}}_P turn\_around, walk\_to\_bus\_stop \overset{at\_university}{\underset{near\_home}{+}}_P walk\_back\_home, get\_on\_bus \overset{at\_university\&bus\_arrived}{\underset{near\_home}{+}}_P turn\_around, get\_on\_bus \overset{at\_university\&bus\_arrived}{\underset{near\_home}{+}}_P walk\_back\_home, stay\_on\_bus \overset{at\_university\&on\_bus}{\underset{near\_home}{+}}_P turn\_around, stay\_on\_bus \overset{at\_university\&on\_bus}{\underset{near\_home}{+}}_P walk\_back\_home, get\_off\_bus \overset{at\_university}{\underset{near\_home}{+}}_P turn\_around, get\_off\_bus \overset{at\_university}{\underset{near\_home}{+}}_P walk\_back\_home\}$
Concurrency relation: $\{\phi\}$
Now, the relationship between *get_on_bus* and *stay_on_bus* turns to be $get\_on\_bus \overset{at\_university\&bus\_arrived\&on\_bus}{\rightsquigarrow}_P stay\_on\_bus$ which is consistent with $get\_on\_bus \overset{at\_university}{\rightsquigarrow}_P stay\_on\_bus$ in the original plan library. Someone may notice the difference of context conditions in these two strict order relationships. However, when *at_university&bus_arrived&on_bus* is true, *at_university* must be true. In other words, $at\_university\&bus\_arrived\&on\_bus \models at\_university$. We will cover behavioural profile entailment in section 3.3.4.

3.3.3 Behavioural Profile Identity

Definition 7 (Behavioural Profile Identity) Behavioural profiles are identical iff $\mathbb{B}_P = \mathbb{B}'_P$. That means for every $x \overset{c1}{\rightsquigarrow}_P y$, $x \overset{c1}{\underset{c1'}{+}}_P y$ or $x \overset{c1}{\|}_P y$ in $\mathbb{B}_P$, there exists a $x \overset{c2}{\rightsquigarrow}_P y$, $x \overset{c2}{\underset{c2'}{+}}_P y$ or $x \overset{c2}{\|}_P y$ in $\mathbb{B}'_P$ such that $c_1 = c_2$ and $c'_1 = c'_2$.

Some cases of behavioural profile identity are listed as follows:
Case 1: Behavioural profiles are identical when adjacent actions or subgoals are repeated:
Plan 1
+!g1 : c ← a_1; a_2; a_3.
Strict Order Relation: $\{a_1 \overset{c}{\rightsquigarrow}_P a_2, a_2 \overset{c}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$
Plan 2
+!g1 : c ← a_1; a_1; a_2; a_3.
Strict Order Relation: $\{a_1 \overset{c}{\rightsquigarrow}_P a_2, a_2 \overset{c}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$

Case 2: Behavioural profiles are identical when additional goal is empty:
Plan 1
+!g1 : c ← a_1; a_2; a_3.
Strict Order Relation: $\{a_1 \overset{c}{\rightsquigarrow}_P a_2, a_2 \overset{c}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$
Plan 2
+!g1 : c ← a_1; a_2; a_3.
+!g2.
Strict Order Relation: $\{a_1 \overset{c}{\rightsquigarrow}_P a_2, a_2 \overset{c}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$

Case 3: Behavioural profiles are identical when goal is different:
Plan 1
+!g1 : c ← a_1; a_2; a_3.
Strict Order Relation: $\{a_1 \overset{c}{\rightsquigarrow}_P a_2, a_2 \overset{c}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$

Plan 2

+!g2 : c ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c}{\rightsquigarrow}_P a_2, a_2 \overset{c}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Case 4: Behavioural profiles are identical when relationships can be transited:

Plan 1

+!g : c ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c}{\rightsquigarrow}_P a_2, a_2 \overset{c}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Plan 2

+!g : c ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c}{\rightsquigarrow}_P a_2, a_2 \overset{c}{\rightsquigarrow}_P a_3, a_1 \overset{c}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Case 5: Behavioural profiles are identical when relationships can be reversed:

Plan 1

+!g : c1 ← a_1; a_2.

+!g : c2 ← a_3; a_4.

Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_3 \overset{c2}{\rightsquigarrow}_P a_4\}$

Exclusiveness Relation: $\{a_1 \underset{c2}{\overset{c1}{+_P}} a_3, a_1 \underset{c2}{\overset{c1}{+_P}} a_4, a_2 \underset{c2}{\overset{c1}{+_P}} a_3, a_2 \underset{c2}{\overset{c1}{+_P}} a_4\}$

Concurrency Relation: $\{\phi\}$

Plan 2

+!g : c2 ← a_3; a_4.

+!g : c1 ← a_1; a_2.

Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_3 \overset{c2}{\rightsquigarrow}_P a_4\}$

Exclusiveness Relation: $\{a_3 \underset{c1}{\overset{c2}{+_P}} a_1, a_3 \underset{c1}{\overset{c2}{+_P}} a_2, a_4 \underset{c1}{\overset{c2}{+_P}} a_1, a_3 \underset{c1}{\overset{c2}{+_P}} a_2\}$

Concurrency Relation: $\{\phi\}$

3.3.4 Behavioural Profile Entailment

Definition 8 (Behavioural Profile Entailment) $\mathbb{B}_P \models \mathbb{B}'_P$ iff. for every $x \overset{c1}{\rightsquigarrow}_P y$, $x \underset{c1'}{\overset{c1}{+}}_P y$ or $x \overset{c1}{\|}_P y$ in $\mathbb{B}_P$, there exists a $x \overset{c2}{\rightsquigarrow}_P y$, $x \underset{c2'}{\overset{c2}{+}}_P y$ or $x \overset{c2}{\|}_P y$ in $\mathbb{B}'_P$ such that $c_1 \models c_2$ and $c'_1 \models c'_2$.

In other words, for every context condition that is true in first behavioural profile, if it is impossible to find a false corresponding context condition in second behavioural profile, then the first behavioural profile entails the seconds behavioural profile. Behavioural profile entailment allows relationships with different context conditions that can be entailed.

Some cases of behavioural profile entailment are listed as follows:

Case 1: Behavioural profiles could be entailed when at least one common element exists in OR joined context conditions:

Plan 1

+!g : c1 ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Plan 2

+!g : c1 | c2 | c3 ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c1|c2|c3}{\rightsquigarrow}_P a_2, a_2 \overset{c1|c2|c3}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

As c1 $\models$ (c1 | c2 | c3), behavioural profile for **Plan 1** entails behavioural profile for **Plan 2**, $\mathbb{B}_{P1} \models \mathbb{B}_{P2}$. Therefore, when c1 is true, agent can perform same behavioural by using either **Plan 1** or **Plan 2**.

Case 2: Behavioural profiles could be entailed when there exists empty context condition:

Plan 1

+!g ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \rightsquigarrow_P a_2, a_2 \rightsquigarrow_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Plan 2

+!g : c1 & c2 ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c1\&c2}{\rightsquigarrow}_P a_2, a_2 \overset{c1\&c2}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Plan 3

+!g : c1 | c2 | c3 ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c1|c2|c3}{\rightsquigarrow}_P a_2, a_2 \overset{c1|c2|c3}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

As (c1 & c2) $\models \phi$, (c1 | c2 | c3) $\models \phi$, behavioural profile for **Plan 2** entails behavioural profile for **Plan 1**($\mathbb{B}_{P2} \models \mathbb{B}_{P1}$) and behavioural profile for **Plan 3** entails behavioural profile for **Plan 1**($\mathbb{B}_{P3} \models \mathbb{B}_{P1}$). What worth mentioning is in this case, **Plan 2** entails **Plan 3**($\mathbb{B}_{P2} \models \mathbb{B}_{P3}$) as well due to **Case 1**.

Case 3: Behavioural profiles could be entailed when **ONLY** duplicate elements occured in AND joined context condition:

Plan 1

+!g : c ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c}{\rightsquigarrow}_P a_2, a_2 \overset{c}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Plan 2

+!g : c & c & c ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c\&c\&c}{\rightsquigarrow}_P a_2, a_2 \overset{c\&c\&c}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Case 4: Behavioural profiles could be entailed when some elements are removed from AND joined context condition:

Plan 1

+!g : c1 & c2 & c3 ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c1\&c2\&c3}{\rightsquigarrow_P} a_2, a_2 \overset{c1\&c2\&c3}{\rightsquigarrow_P} a_3\}$

Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$
Plan 2
+!g : c1 & c2 ← a_1; a_2; a_3.
Strict Order Relation: $\{a_1 \overset{c1\&c2}{\rightsquigarrow}_P a_2, a_2 \overset{c1\&c2}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$

3.3.5 Behavioural Preservation

Examples in section 3.3.2 reveals processes of finding conflicts in agent behavioural profile and method of resolving conflicts. Generally speaking, if there are no conflicts in behavioural profiles, then agent behaviour is preserved after redesign. In order to well define the agent behavioural preservation, we need a formal definition of behavioural preservation.

Definition 9 (Behavioural Preservation) Behavioural of an agent is preserved if and only if $\mathbb{B}_P \subseteq \mathbb{B}'_P$ or $\mathbb{B}_P \models \mathbb{B}'_P$, where $\mathbb{B}_P$ is the original agent behavioural profile and $\mathbb{B}'_P$ is the redesigned agent behavioural profile.

That means if all relationships $\{\rightsquigarrow_P, +_P, \|_P\}$ defined in original agent plan library still hold after agent plan being redesigned, agent behaviour can be regarded to be preserved. What worth mentioning is definition of relation transitivity and relation reversibility can be applied in checking behavioural preservation. Under the condition that relations defined in original agent behavioural profile can be implied from redesigned agent behavioural profile, those relationships are regarded as identical as well. For instance, if $\{x \overset{c}{\rightsquigarrow}_P z\} \subseteq \rightsquigarrow_P$ and $\{x \overset{c}{\rightsquigarrow}_P y, y \overset{c}{\rightsquigarrow}_P z\} \subseteq \rightsquigarrow'_P\}$, then $\{x \overset{c}{\rightsquigarrow}_P z\}$ and $\{x \overset{c}{\rightsquigarrow}_P y$ and $y \overset{c}{\rightsquigarrow}_P z\}$ are identical.

Some cases of behavioural profile preservation are listed as follows:

Case 1: Behaviour could be preserved when actions are moved to sub-goals:

Plan 1
+!g : c1 ← a_1; a_2; a_3.
Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Plan 2

+!g : c1 ← a_1; !sg_2; a_3.

+!sg2 : c2 ← a_2.

Strict Order Relation: $\{a_1 \overset{c1\&c2}{\rightsquigarrow}_P a_2, a_2 \overset{c1\&c2\&c1}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

As $c1\&c2 \models$ c1 and $c1\&c2\&c1 \models$ c1, behavioural profile for Plan 2 entails behavioural profile for Plan 1, $\mathbb{B}_{P2} \models \mathbb{B}_{P1}$. Consequently, behaviours in Plan 1 are preserved in Plan2.

Case 2: Behaviour could be preserved when additional actions are introduced:

Plan 1

+!g : c1 ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Plan 2

+!g : c1 ← a_1; a_2; a_3; a_4.

Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3, a_3 \overset{c1}{\rightsquigarrow}_P a_4\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

As behavioural profile for Plan 1 is a subset of behavioural profile for Plan 2, Plan 2 preserve original behaviour in Plan 1.

Case 3: Behaviour could be preserved when additional subgoals are introduced:

Plan 1

+!g : c1 ← a_1; a_2; a_3.

Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3\}$

Exclusiveness Relation: $\{\phi\}$

Concurrency Relation: $\{\phi\}$

Plan 2

+!g : c1 ← a_1; a_2; a_3; !sg_4.

$+!sg_4$: c2 ← a_4.
Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3, a_3 \overset{c1}{\rightsquigarrow}_P !sg_4, !sg_4 \overset{c2}{\rightsquigarrow}_P a_4\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$
As behavioural profile for Plan 1 is a subset of behavioural profile for Plan 2, Plan 2 preserve original behaviour in Plan 1.

Case 4: Behaviour could be preserved when additional goals are introduced:
Plan 1
+!g : c1 ← a_1; a_2; a_3.
Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$
Plan 2
+!g : c1 ← a_1; a_2; a_3.
+!ga : c2 ← a_4.
Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3, a_4 \underset{c2}{\overset{c2}{+}}_P a_4\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$
As behavioural profile for Plan 1 is a subset of behavioural profile for Plan 2, Plan 2 preserve original behaviour in Plan 1.

3.3.6 Relationships between Behavioural Preservation, Identity and Entailment

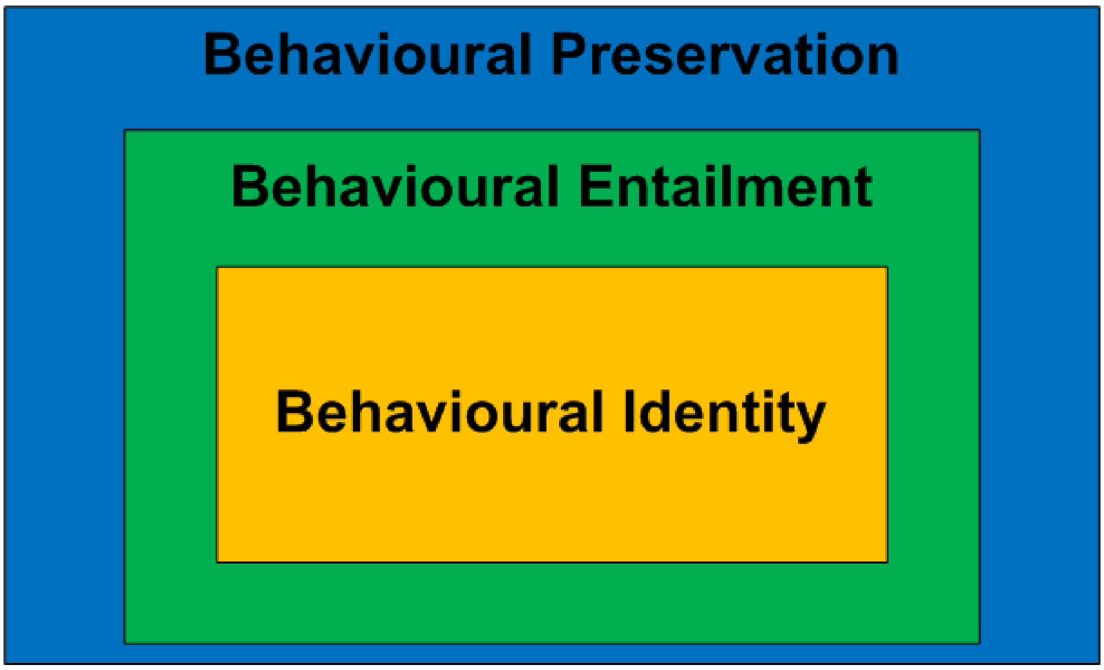


Figure 3.11: Relationships between Behavioural Preservation, Identity and Entailment

If behavioural profiles are identical, those behavioural profiles can be mutually entailed. Because context conditions in those behavioural profiles are exactly the same. For those behavioural profiles that can be entailed, entailed agent can still perform behaviours of original agent. Therefore, agent behaviours are preserved. We can thus arise the summary that agent behavioural preservation includes behavioural entailment, behavioural entailment includes behavioural identity. A figure in 3.11 shows relationship among them vividly.

3.4 Preserve Agent Behaviours while Increasing Agent Reactivity

As known in section 3.1, factors affecting agent reactivities are sub-goals decorated with context conditions, actions with context conditions in front as well as fresh new plans for other event. Consequently, it might not possible to increase agent reactivity in the scope of behavioural profile identity and behavioural profile entailment. The reason is, when behaviour profiles are identical or can be entailed, new sub-goals or plans are not allows. Hence, agent behavioural-preserved reactivity enhancement can only be performed in the scope of behavioural preservation $\oplus$ behavioural entailment. Analysing behavioural preservation cases in section 3.3.5, only Case 1 and Case 4 get reactivity enhanced.

Case 1: Behaviour could be preserved when actions are moved to sub-goals:
Plan 1
+!g : c1 $\leftarrow a_1; a_2; a_3$.
Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$

$$R(P1) = 0.5 \times \frac{1}{2+1} + 0.5 \times \frac{0}{0+1} = \frac{1}{6}$$

Plan 2
+!g : c1 $\leftarrow a_1; !sg_2; a_3$.
+!sg2 : c2 $\leftarrow a_2$.
Strict Order Relation: $\{a_1 \overset{c1\&c2}{\rightsquigarrow}_P a_2, a_2 \overset{c1\&c2\&c1}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$

$$R(P1) = 0.5 \times \frac{1}{1+1} + 0.5 \times \frac{1}{0+1} = \frac{3}{4}$$

Case 4: Behaviour could be preserved when additional goals are introduced:
Plan 1
+!g : c1 $\leftarrow a_1$; a_2; a_3.
Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$

$$R(P1) = 0.5 \times \frac{1}{2+1} + 0.5 \times \frac{0}{0+1} = \frac{1}{6}$$

Plan 2
+!g : c1 $\leftarrow a_1$; a_2; a_3.
+!ga : c2 $\leftarrow a_4$.
Strict Order Relation: $\{a_1 \overset{c1}{\rightsquigarrow}_P a_2, a_2 \overset{c1}{\rightsquigarrow}_P a_3, a_4 \overset{c2}{\underset{c2}{+}}_P a_4\}$
Exclusiveness Relation: $\{\phi\}$
Concurrency Relation: $\{\phi\}$

$$R(P1) = (0.5 \times \frac{1}{2+1} + 0.5 \times \frac{0}{0+1}) + (0.5 \times \frac{1}{0+1}) = \frac{2}{3}$$

Therefore, in order to preserve agent behaviour while enhancing agent reactivity, we can move actions to sub-goals or create fresh new plans.

Chapter 4

Evaluation

In order to evaluate measures and algorithms in this thesis, a prototype is implemented and several experiments are described and discussed in this chapter.

4.1 Implementation

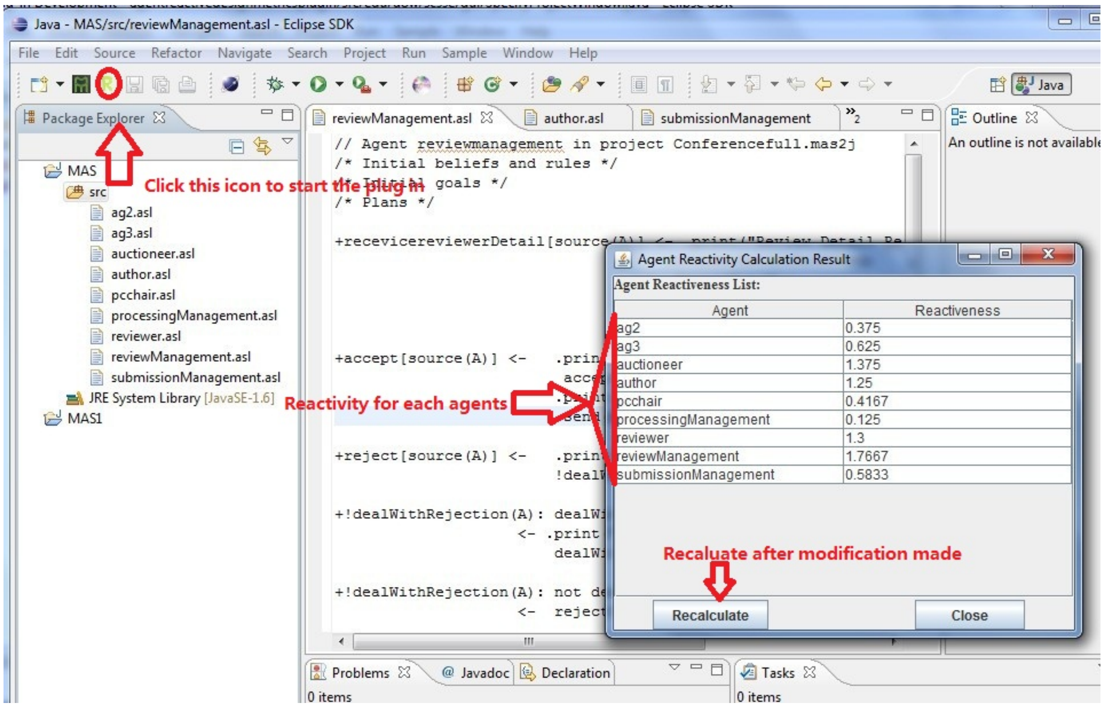
Based on the reactivity measure for BDI agent programs and the algorithm to calculate it from agents' plan libraries, a prototype has been developed to demonstrate the effectiveness of the reactivity measure. More specifically, a plugin[1] has been implemented for the Eclipse-based code editor of Jason[2], one of the most well-known platforms for developing agent applications in AgentSpeak[3].

The reactivity plugin reads the source code of an agent program, which may consist of a number of agents. The plugin analyses the plan library of each agent in the program and constructs the graphs representing the domain model of the top triggering event in the plan library. It then calculates the reactivity of each agent's plan library using the algorithm described in the previous section. Figure 4.1 shows a snapshot of the reactivity plugin for Jason. Having the plugin installed, Jason developers can invoke it by clicking on a small icon namely R (R for reactivity) on the toolbar of the Eclipse-based editor for Jason. A small

[1]The plugin is available at `http://designmetrics.googlecode.com/svn/designmetrics/`

[2]`http://jason.sourceforge.net/Jason/`

[3]AgentSpeak is one of the most well-known languages for implementing BDI agent.


Java - MAS/src/reviewManagement.asl - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Sample Window Help
Package Explorer
MAS
src
ag2.asl
ag3.asl
auctioneer.asl
author.asl
pcchair.asl
processingManagement.asl
reviewer.asl
reviewManagement.asl
submissionManagement.asl
JRE System Library [JavaSE-1.6]
MAS1
Click this icon to start the plug in
reviewManagement.asl
author.asl
submissionManagement
Outline
An outline is not available
Reactivity for each agents
Agent Reactivity Calculation Result
Recaluate after modification made
Recalculate
Close
Problems
Javadoc
Declaration
0 items
Tasks
0 items

```
// Agent reviewmanagement in project Conferencefull.mas2j
/* Initial beliefs and rules */
/* Initial goals */
/* Plans */

+recevicereviewerDetail[source(A)] <-   .print("Review Detail Re

+accept[source(A)] <-    .prin
                          acc
                         .p
                          sen

+reject[source(A)] <-    .prin
                         !dealW

+!dealWithRejection(A): dealWi
                    <- .print
                       dealWi

+!dealWithRejection(A): not de
                    <-  reject
```

Agent Reactiveness List:

Agent	Reactiveness
ag2	0.375
ag3	0.625
auctioneer	1.375
author	1.25
pcchair	0.4167
processingManagement	0.125
reviewer	1.3
reviewManagement	1.7667
submissionManagement	0.5833

Figure 4.1: A snapshot for the reactivity plugin for the Eclipse-based Jason editor

dialog would be displayed with the reactivity value for each agent in the current program. The developers can modify the program and the reactivity values may be updated to reflect the changes.

4.2 Preliminary empirical study

Using the prototype tool technique has been evaluated by conducting a preliminary empirical study. In the small experiment, an investigation is done on the following research question: *programs showing a high value in the reactivity measure perform better in responding to changes in the environment than those with a low value.*

In order to assess the hypothesis, an experiment has been performed on a simple agent-based auction system that has been developed with the inspiration of [9]. There are four agents in this system: an auctioneer and three bidders. Three different sets of environments have been designed. First, the static environment generates only one type of event indicating when the auction starts. Second, the relatively dynamic environment generates two types of events indicating when the auction starts and when bids are accepted. Third, the extremely dynamic environment generates not only the previous two event types but also the event when a bid is successful. The three bidders are classified into three levels: basic, intermediate and advanced. The basic bidder has a plan to handle the event of an auction starting by submitting a bid (as a primitive action). The intermediate bidder also has a plan to handle such event but submitting a bid is designed as a subgoal which is handled by another plan that is applicable only when a permission for bidding is broadcast. Finally, the advanced bidder has the same plans as in the intermediate bidder and a plan to handle a successful bidding. The reactivity tool was used to calculate the reactivity of the three types of agent. The reactivity value of the basic bidder is 0.2, the intermediate bidder is 0.5, and the advanced bidder is 1.1.

Agent/Environment	Basic	Intermediate	Advanced
Static	1	1	1
Relatively dynamic	1	2	2
Extremely dynamic	1	2	3

Figure 4.2: The scores of three bidders in three types of environment

The auction system was run in the three types of environments described earlier to observe behaviours of three bidders. In order to assess how reactive the agents are during execution, a rough score is given based on how their behaviour changes when a significant event occurs in the environment. More specifically, assume that there are n significant events potentially occurring in the environment (e.g. $n = 3$ for the extremely dynamic environment in the experiment). The behaviour of an agent during execution is observed and every time its behaviour changes to respond to an event occur in the environment, one mark is given. The score for each bidder in each type of environment is shown in Figure 4.2.

As can be seen, the basic bidder gets the same score (i.e. 1) in all three types of environment. In addition, for the static environment all the three bidders score the same mark. The advanced bidder however has the highest score (i.e. 3) in the extremely dynamic environment.

As mentioned earlier, the reactivity measure indicates the advanced bidder is the most reactive agent while the basic agent is the least reactive one. Results from the experiments by executing those agents confirm this in which the advanced agent outperforms the other agents (with respect to reactivity) in the extremely dynamic environment.

4.3 Experiment on subgoals

Another small experiment has also been conducted to investigate on the research question: *subgoals with context conditions can enhance the reactivity of the agent.*

In this experiment, a state transition system was set up to describe different behaviours of the system. The time-based state transition system produces events in a certain period of time. For experimental purposes, two agent plan library profiles were set up. The first library(reactivity value of 0.1) has a plan with all atomic actions, and the second library (reactivity value of 0.5) has a number of plans and subgoals as follows:

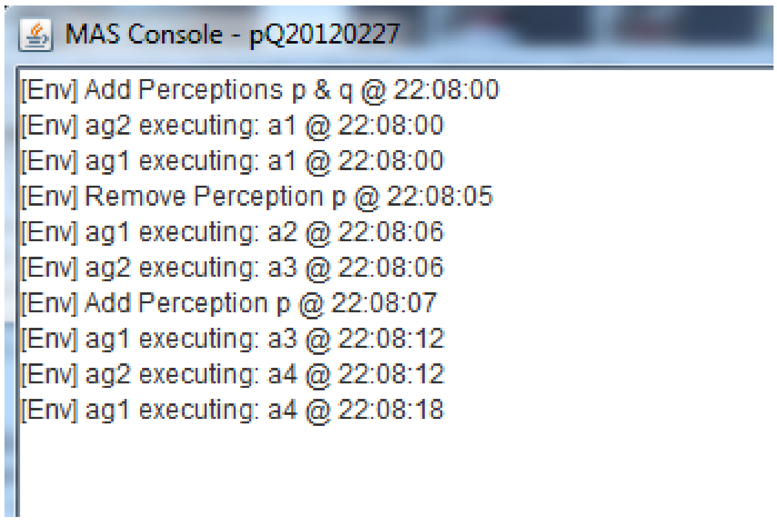
```
!start.
+!start : p & q ← a1; a2; a3; a4.
```

Figure 4.3: Plan Library for Agent 1

```
!start.
+!start : p & q ← a1; !sg2; a3; !sg4.
+!sg2: p & q ← a2.
+!sg2.
+!sg4: p & q ← a4.
+!sg4.
```

Figure 4.4: Plan Library for Agent 2

Initially (at t_0), the environment is set up such that both p and q are true. After 5 seconds (i.e. at t_0 +5), the environment changes such that p is no longer true and after 7 seconds (i.e. at $t_0 + 7$), the environment changes again such that p is true again. Assuming every action in agents takes 6 seconds to complete, the experiment is then run for 24 seconds and log the execution of both agents. System logging(figure 4.5) of an execution of the agent system is attached as follows:


MAS Console - pQ20120227

```
[Env] Add Perceptions p & q @ 22:08:00
[Env] ag2 executing: a1 @ 22:08:00
[Env] ag1 executing: a1 @ 22:08:00
[Env] Remove Perception p @ 22:08:05
[Env] ag1 executing: a2 @ 22:08:06
[Env] ag2 executing: a3 @ 22:08:06
[Env] Add Perception p @ 22:08:07
[Env] ag1 executing: a3 @ 22:08:12
[Env] ag2 executing: a4 @ 22:08:12
[Env] ag1 executing: a4 @ 22:08:18
```

Figure 4.5: Agent system logging

The execution log of the agent with the first plan library indicates that it has performed (a1; a2; a3; a4), regardless of changes in the environment with respect to context condition p. Meanwhile, with the second plan library the agent has performed(a1; a3; a4), indicating the agent has responded to changes in the environment. Five seconds after performing action a1, context condition p was no longer true and the execution of subgoal sg2 resulted in no action. At 22:08:07 (the deliberation of checking context condition took 1 second), context condition p was true again and the execution of subgoal sg4 resulted in the execution of action a4.

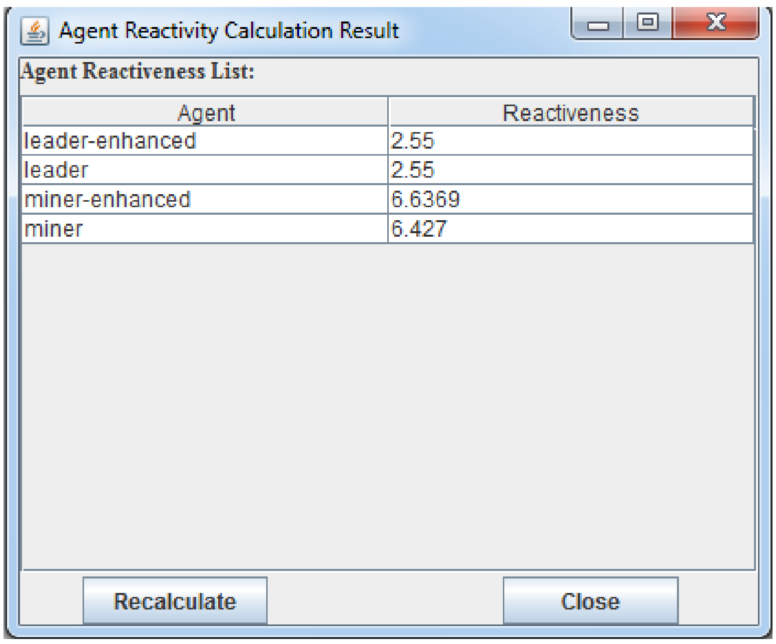
When context condition p was no longer true, agent 2 did not perform action a2. However, agent 1 performed a2, which was not as accurate as agent 2. The experiment proves that agent 2, which has more subgoals (with context conditions) than agent 1, is more reactive than agent 1. However, in this experiment, action a4 does not help in comparing agent 1 and agent 2 plan libraries. It might be better to add and remove event p or q at different time to provide more solid evidence that subgoals with context conditions can improve the agent reactivity.

4.4 Experiment on external events

A complex experiment has been conducted based on the gold miner example in the Jason agent platform. This experiment is used to investigate the following research question: *adding plans for external events can enhance the reactivity of the agent.*

The experiment is made in static and dynamic environments respectively. In the original version of gold miner, miners can detect gold, picking gold, sharing position of gold they detected and giving up the belief of picking gold if it has been picked by other miners. It cannot be denied that the example is robust on cooperation between miners and exception handling. However, in order to do a comparison, an enhanced version of gold miner agent program is developed. In the enhanced version of gold miner agent program, apart from handling gold, miners can get the perception of diamonds and pick diamonds.

In static environment, the plugin in eclipse that developed based on the reactivity measure is used to calculate reactivity of all agents in gold miners program. In the experiment, weights for triggering events $\{\alpha_i\}$ are assumed to be identical to each other $\{\alpha_i = 1 \mid \text{i} \in [0, \text{n}]\ \}$.Weight of reactivity to external events k is assumed to be 0.5. After running the plugin, the reactiveness of original gold-only miner agent is 6.427, while the reactiveness of enhanced gold-and-diamond miner agent climbs to 6.6369 as shown in Figure 4.6. As leader agent plan libraries are the same, reactivity of both leader plan libraries are 2.55. As can be seen from the statistics in this experiment, the systems reactiveness increased due to the enhancement.

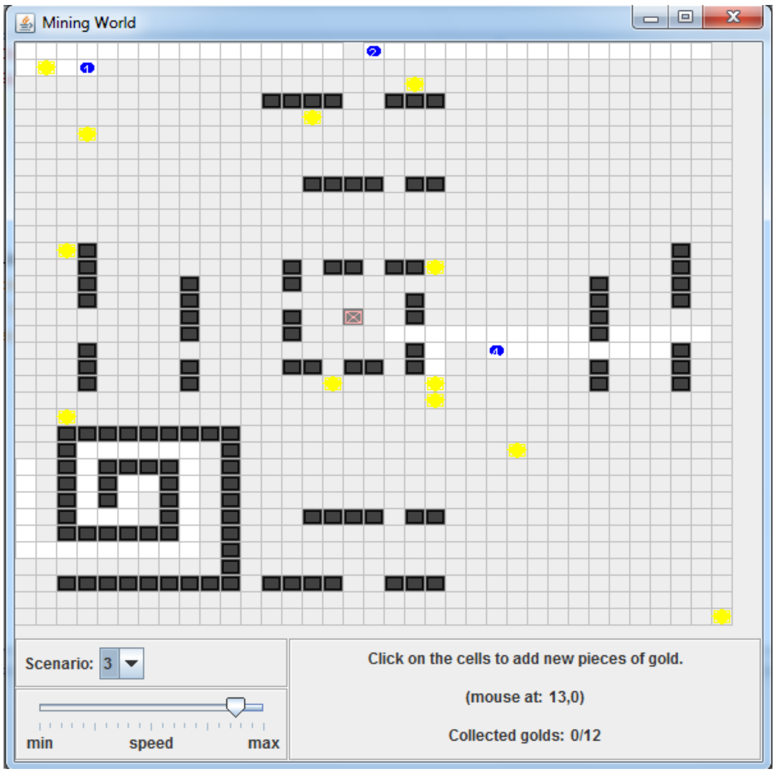


Agent Reactiveness List:

Agent	Reactiveness
leader-enhanced	2.55
leader	2.55
miner-enhanced	6.6369
miner	6.427

Figure 4.6: Gold miner multi-agent system reacitivity

In dynamic environment, perception of diamond is added into miners with plan library that can deal with gold only. By running the gold miners system 4.7, miners do not make any response to diamonds. In comparison, agent system with enhanced plan library is executed. This time, miners can pick both gold and diamond shown in figure 4.8. In addition, when miner in left-bottom corner is on its way to right-bottom corner for picking a diamond, if the diamond is picked by miner at the right-bottom corner, the miner at left-bottom corner goes back to left-bottom corner and keeps researching for resources. Results from dynamic environment experiment provide evidence that reactivity of the system get increased by adding plans for external events.

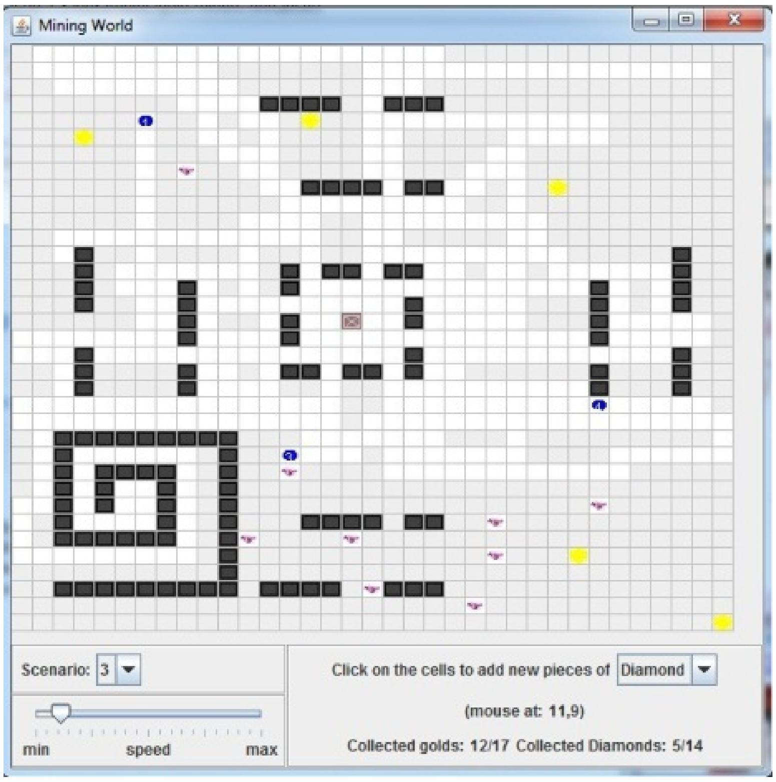


Click on the cells to add new pieces of gold.

(mouse at: 13,0)

Collected golds: 0/12

Figure 4.7: Gold miners for picking gold only



Click on the cells to add new pieces of Diamond

(mouse at: 11,9)

Collected golds: 12/17 Collected Diamonds: 5/14

Figure 4.8: Gold miners for picking gold and diamond

This experiment has been conducted in static as well as dynamic environments. When it comes to the static environment, adding plans for external event can trigger an increase on the reactivity measure result. It proves that the measure works for plans for external events. In the dynamic environment, it can be observed that enhanced gold miners can make reactions on diamonds while the original version of gold miners cannot. It not only proves the conclusion made in the static environment is correct, but also proves adding plans for external events can increase the reactivity of the agent. This experiment proves that adding plans for one extra external event can enhance the reactivity of the agent. However, it might be better if more experiments on different number of external events can be conducted.

4.5 Experiment on a paper submission multi-agent system

In addition, a large experiment has been conducted to assess influences from both external events and subgoals. In paper submission processes, author submits the paper to a submission management system. The submission management system generates paper ID and stores paper before sending an acknowledgement message back to the client. After submission due date, PC chair enters reviewer details into the review management system. Then, the reviewer management system would invite reviewers. On receiving acceptance from reviewers, the review management system asks reviewers for preference. Then it gets the preference and assigns papers to corresponding reviewer. Reviewers then review papers and send result back to the review management system. The review management system then collects all the review reports. After review deadline, PC Chair will make decision based on review reports and send his decision to review management system. The Review management system then finalizes decision and sends notification back to authors. If author receives acceptance, a camera-ready will be sent to process management system. The process management system then collects camera-ready and sends to publisher. Based on activities in the system, a multi-agent system with environment 4.1 and agents plan libraries 4.2, 4.3, 4.4, 4.5 4.6, 4.7 are created to simulate conference paper submission process.

Listing 4.1: Environment for Paper Submission Multi-Agent System

```java
import jason.asSyntax.*;
import jason.environment.*;
import java.util.logging.*;
import java.util.Random;
public class ConEnv extends Environment {
  private final static int numOfAuthors = 4;
  private Logger logger = Logger.getLogger("Conferencefull.mas2j."
      + ConEnv.class.getName());
  private Integer paperReceived  = null;
  private Random r = new Random(System.currentTimeMillis());
  private boolean invited[] = new boolean[11];
  private boolean paperselected[] = new boolean[100];
```

```java
/** Called before the MAS execution with the args informed in
    .mas2j */
@Override
public void init(String[] args) {
  super.init(args);
  paperReceived = new Integer(0);
  for (int i = 0; i < 10; i ++) {
    invited[i] = false;
  }
  for (int i = 0; i < 100; i++) {
    paperselected[i] = false;
  }
  addPercept(Literal.parseLiteral("hasPaper(\"Paper\")"));
  int rd = Math.abs(r.nextInt()) % 101;
  if (rd > 50) {
    addPercept("submissionManagement",
        Literal.parseLiteral("databaseAvailable"));
  }
  rd = Math.abs(r.nextInt())% 101;
  if (rd > 40) {
    addPercept("submissionManagement",
        Literal.parseLiteral("internetAccess"));
  }
  addPercept("reviewManagement",
      Literal.parseLiteral("dealWithRejection"));
}
@Override
public boolean executeAction(String agName, Structure action) {
  int rd = Math.abs(r.nextInt()) % 101;
  if(action.getFunctor().equals("storePaper")) {
    paperReceived ++;
    addPercept("reviewManagement",
        Literal.parseLiteral("paperNum("+ paperReceived +")"));
  } else if(action.getFunctor().equals("storePaperLocally")) {
    paperReceived ++;
    addPercept("reviewManagement",
        Literal.parseLiteral("paperNum("+ paperReceived +")"));
  } else if (action.getFunctor().equals("paperReceived")) {
    if(rd > 50) {
      addPercept(agName, Literal.parseLiteral("emergency"));
      }
      try {
        Thread.sleep(200);
      } catch (Exception e) {}
    }
    else if(action.getFunctor().equals("inviteReviewers")) {
      int paperNum = (int)((NumberTerm)action.getTerm(0)).solve();
      for (int i = 0; i < paperNum; i++) {
        int reviewer = Math.abs(r.nextInt()) % 10 + 1;
        if(invited[reviewer - 1]) {
          i--;
          continue;
        }
        addPercept("reviewer"+reviewer,
            Literal.parseLiteral("invitationReceived"));
        invited[reviewer - 1] = true;
      }
    }
    else if(action.getFunctor().equals("invitationReceived")) {
      if(rd > 20 ) {
        addPercept(agName, Literal.parseLiteral("accept"));
      }
    } else if(action.getFunctor().equals("sendReviewPref")) {
      for(int i = 0; i < paperReceived; i++) {
        if(!paperselected[i]) {
          paperselected[i] = true;
          addPercept("reviewManagement",
              Literal.parseLiteral("receivePrefs("+agName+","+(i+1)+")"));
          break;
        }
      }
    } else if(action.getFunctor().equals("sendNotification")) {
      for(int i = 1; i <= numOfAuthors; i++) {
        int temp = (Math.abs(r.nextInt()) % 101);
        if( temp > 50) {
          addPercept("author"+i,
              Literal.parseLiteral("availableTOGetResult"));
        }
        try {
          Thread.sleep(200);
```

```
            } catch (Exception e) {}
        }

        int paperNum = (int)((NumberTerm)action.getTerm(0)).solve();
        for(int i = 1; i <= paperNum; i++) {
          rd = Math.abs(r.nextInt())% 101;
          if(rd > 50) {
            addPercept("author"+i,  Literal.parseLiteral("accept"));
          } else {
            addPercept("author"+i,  Literal.parseLiteral("reject"));
          }
        }
      } else if(action.getFunctor().equals("dealWithReject")) {
        for(int i = 0; i < 1; i++) {
          int reviewer = Math.abs(r.nextInt())% 10 + 1;
          if(invited[reviewer - 1]) {
            i--;
            continue;
          }
          addPercept("reviewer"+reviewer,
              Literal.parseLiteral("invitationReceived"));
          invited[reviewer-1] = true;
        }
      } else
          if(action.getFunctor().equals("inviteReviewersInEmergency"))
          {
        /*if not all the reviewer are invited*/
        boolean flag = false;
        for(int i = 0; i < 10; i++) {
          if(invited[i] == false) {
            flag = true;
          }
        }
        if(!flag) {
          logger.warning("No Reviewer is available!");
          return false;
        }
        int paperCode = (int)((NumberTerm)action.getTerm(0)).solve();
        paperselected[paperCode - 1] = false;
        for(int i = 0; i < 1; i++) {
            int reviewer = Math.abs(r.nextInt()) % 10 + 1;
            if(invited[reviewer - 1]) {
              i--;
              continue;
            }
            addPercept("reviewer"+reviewer,
                Literal.parseLiteral("invitationReceived"));
            invited[reviewer -1] = true;
          }

      }
      return true;
    }
    /** Called before the end of MAS execution */
    @Override
    public void stop() {
      super.stop();
    }
}
```

Listing 4.2: Less reactive author plan library(Reactivity 0.9583)

```
/* Initial goals */
!submit_Paper.
/* Plans */
+!submit_Paper:hasPaper(P) <- ?hasPaper(P);
                              .print("Submit Paper");
                              .send(submissionManagement,tell,receivePaper(P)).
+!submit_Paper:not hasPaper(P).
+receiveAck[source(A)]<- .print("Ack received");
                          ackReceived(A).
+accept[source(A)] <- .print("Receive Accept");
                       receiveAccept;
                      .print("Prepare Camera-ready");
                      prepareCameraReady;
                      .print("Send Camera-read");
                      .send(processingManagement, tell,
                         receiveCameraReady);
                      -accept.
+reject[source(A)] <- .print("Receive Reject");
```

```
                          receiveReject;
                          -reject.
```

Listing 4.3: Less reactive PC chair plan library(Reactivity 0.4167)

```
/* Initial goals */
!wait.
/* Plans */
+!wait <- .wait(3000);
         +submissionDeadlinePassed;
         !enterReviewerDetail.
+!enterReviewerDetail:submissionDeadlinePassed
                        <-  .print("Enter Reivewer Detail");
                            .send(reviewManagement, tell,
                               recevicereviewerDetail);
                            .wait(6000);
                            .print("Read Review");
                            readReview;
                            .print("Make a Decision");
                            .send(reviewManagement, tell,
                               receiveDecision).
+!enterReviewerDetail: not submissionDeadlinePassed
                      <- .wait(100);
                         !enterReviewerDetail.
```

Listing 4.4: Less reactive submission managememt agent plan library(Reactivity 0.1667)

```
/* Plans */
+receivePaper(P)[source(A)] <- .print(P," Received From ",A);
                               paperReceived(A,P);
                               !generatePaperID(A,P);
                               !storePaper(A,P);
                               .print("Send Ack to ", A);
                               .send(A, tell, receiveAck).
+!generatePaperID(A,P)<- .print("Generate Paper ID For ", A , "'s
    ", P);
                         generatePaperID(A,P).
+!storePaper(A,P) <- .print("Store Paper For ", A, "'s ", P);
                      storePaper(A,P).
```

Listing 4.5: Less reactive process management agent plan library(Reactivity 0.125)

```
/* Plans */
+receiveCameraReady[source(A)] <- .print("Camera-ready Received
    from ", A);
                                 cameraReadyReceived;
                                 .print("Collect Camera-ready");
                                 collectCameraReady;
                                 .print("Send to Publisher");
                                 sendToPublisher.
```

Listing 4.6: Less reactive review management agent plan library(Reactivity 0.9333)

```
/* Plans */

+recevicereviewerDetail[source(A)] <- .print("Review Detail Recived
    From ", A);
                                     reviewerDetailRecevied;
                                     ?paperNum(N);
                                     .print("Invite Reviewers");
                                     inviteReviewers(N).
+accept[source(A)] <- .print("Receive Accept to Review From ", A);
                      acceptToReview;
                      .print("Ask ",A," For Prefs ");
                      .send(A, tell, receivePref).
+reject[source(A)] <- .print("Receive Reject to Review From ", A);
                     rejectToReview.
+receivePrefs(R,N)[source(A)] <- .print("Prefs Received From ",R,"
    On Author",N,"'s Paper");
                                preferenceReceived;
                                .print("Collect Prefs");
                                collectPreference;
                                .print("Assign Papers to ", R);
                                .send(R, tell, receviePaper(N)).

+receiveReviewrReports(N)[source(A)] <- .print("Review Reports
    Recivied On Paper",N," From ", A);
                                        receiveReviewReport;
```

```
                                          .print("Collect Review
                                             Reports On Paper", N);
                                          collectReviewReport.
+receiveDecision[source(A)] <- .print("Decision Received");
                                 decisionReceived;
                                 .print("Finalise decision");
                                 finaliseDecision;
                                 .print("Send Notification");
                                 ?paperNum(N);
                                 .print("Send Notification");
                                 sendNotification(N).
```

Listing 4.7: Less reactive reviewer plan library(Reactivity 0.85)

```
/* Initial goals */
+invitationReceived[source(A)]<-.print("Invitation Received");
                                  invitationReceived;
                                  !sendReply.
+!sendReply: accept
            <-  .print("Send Accept to review");
                .send(reviewManagement,tell, accept);
               -accpet.
+!sendReply :not accept
            <-  .print("Send Reject to review");
                .send(reviewManagement, tell, reject).
+receivePref[source(R)] <-  .print("Prefs request received From ",
   R);
                             prefsRequstReceived;
                            .print("Send Prefs");
                             sendReviewPref.
+receviePaper(N)[source(R)] <- .print("Paper ",N," Received");
                                 paperReceived;
                                 .print("Review Paper", N);
                                 reviewPaper;
                                 print("Send Review Reports on
                                    Paper", N);
                                 .send(reviewManagement, tell,
                                    receiveReviewrReports(N)).
```

Listing 4.8: Trace of System Execution

```
[author4] Submit Paper
[author2] Submit Paper
[author1] Submit Paper
[author3] Submit Paper
[submissionManagement] Paper Received From author4
[submissionManagement] Paper Received From author2
[submissionManagement] Paper Received From author1
[submissionManagement] Paper Received From author3
[submissionManagement] Generate Paper ID For author4's Paper
[submissionManagement] Generate Paper ID For author1's Paper
[submissionManagement] Generate Paper ID For author2's Paper
[submissionManagement] Generate Paper ID For author3's Paper
[submissionManagement] Store Paper For author4's Paper
[submissionManagement] Store Paper For author1's Paper
[submissionManagement] Store Paper For author2's Paper
[submissionManagement] Send Ack to author4
[submissionManagement] Store Paper For author3's Paper
[submissionManagement] Send Ack to author1
[author4] Ack received
[submissionManagement] Send Ack to author2
[submissionManagement] Send Ack to author3
[author1] Ack received
[author2] Ack received
[author3] Ack received
[pcchair] Enter Reivewer Detail
[reviewManagement] Review Detail Recived From pcchair
[reviewManagement] Invite Reviewers
[reviewer4] Invitation Received
[reviewer9] Invitation Received
[reviewer6] Invitation Received
[reviewer5] Invitation Received
[reviewer6] Send Accept to review
[reviewer5] Send Accept to review
[reviewer9] Send Reject to review
[reviewer4] Send Reject to review
[reviewManagement] Receive Accept to Review From reviewer6
[reviewManagement] Ask reviewer6 For Prefs
[reviewManagement] Receive Accept to Review From reviewer5
```

```
[reviewer6] Prefs request received From reviewManagement
[reviewer6] Send Prefs
[reviewManagement] Receive Reject to Review From reviewer9
[reviewManagement] Receive Reject to Review From reviewer4
[reviewManagement] Ask reviewer5 For Prefs
[reviewManagement] Prefs Received From reviewer6 On Author1's Paper
[reviewManagement] Collect Prefs
[reviewer5] Prefs request received From reviewManagement
[reviewManagement] Assign Papers to reviewer6
[reviewer5] Send Prefs
[reviewer6] Paper 1 Received
[reviewer6] Review Paper1
[reviewManagement] Prefs Received From reviewer5 On Author2's Paper
[reviewManagement] Collect Prefs
[reviewManagement] Review Reports Recivied On Paper1 From reviewer6
[reviewManagement] Assign Papers to reviewer5
[reviewManagement] Collect Review Reports On Paper1
[reviewer5] Paper 2 Received
[reviewer5] Review Paper2
[reviewManagement] Review Reports Recivied On Paper2 From reviewer5
[reviewManagement] Collect Review Reports On Paper2
[pcchair] Read Review
[pcchair] Make a Decision
[reviewManagement] Decision Received
[reviewManagement] Finalise decision
[reviewManagement] Send Notification
[reviewManagement] Send Notification
[author2] Receive Reject
[author4] Receive Accept
[author1] Receive Reject
[author4] Prepare Camera-ready
[author3] Receive Accept
[author4] Send Camera-read
[author3] Prepare Camera-ready
[author3] Send Camera-read
[processingManagement] Camera-ready Received from author4
[processingManagement] Collect Camera-ready
[processingManagement] Camera-ready Received from author3
[processingManagement] Send to Publisher
[processingManagement] Collect Camera-ready
```

```
[processingManagement] Send to Publisher
```

By enhancing reactivity of agents using the rules defined in section 3.1, agent plan libraries can be redesigned as in 4.9, 4.10, 4.11, 4.12 4.13, 4.14 to receive more perception from environment and deal events more accurately.

Listing 4.9: More reactive author plan library(Reactivity 1.25)

```
/* Initial goals */
!submit_Paper.
/* Plans */
+!submit_Paper:hasPaper(P) <- ?hasPaper(P);
                              .print("Submit Paper");
                              .send(submissionManagement,tell,receivePaper(P)).
+!submit_Paper:not hasPaper(P).
+receiveAck[source(A)]<- .print("Ack received");
                         ackReceived(A).
+accept[source(A)] <- !receiveAccept;
                      .print("Prepare Camera-ready");
                      prepareCameraReady;
                      .print("Send Camera-read");
                      .send(processingManagement, tell,
                         receiveCameraReady);
                      -accept.
+!receiveAccept: availableTOGetResult
                 <- .print("Receive Accept");
                    receiveAccept.

+!receiveAccept:not availableTOGetResult
                 <- .print("<<<Wait to be available>>>");
                    .wait(100);
                    .print("Receive Accept");
                    receiveAccept.

+reject[source(A)] <- .print("Receive Reject");
                      receiveReject;
                      -reject.
```

Listing 4.10: More reactive PC chair plan library(Reactivity 0.4167)

```
/* Initial goals */
!wait.
```

```
/* Plans */
+!wait <- .wait(3000);
          +submissionDeadlinePassed;
          !enterReviewerDetail.
+!enterReviewerDetail:submissionDeadlinePassed
                        <-.print("Enter Reivewer Detail");
                          .send(reviewManagement, tell,
                             recevicereviewerDetail);
                          .wait(6000);
                          .print("Read Review");
                          readReview;
                          .print("Make a Decision");
                          .send(reviewManagement, tell,
                             receiveDecision).
+!enterReviewerDetail: not submissionDeadlinePassed
                      <-.wait(100);
                         !enterReviewerDetail.
```

Listing 4.11: More reactive submission managememt agent plan library(Reactivity 0.5833)

```
/* Plans */
+receivePaper(P)[source(A)] <-  .print(P," Received From ",A);
                                 paperReceived(A,P);
                                 !generatePaperID(A,P);
                                 !storePaper(A, P);
                                 !sendAckToAuthor(A).
+!generatePaperID(A, P) <-  .print("Generate Paper ID For ", A ,
    "'s ", P);
                             generatePaperID(A,P).
+!storePaper(A, P): databaseAvailable
                  <-  .print("Store Paper For ", A, "'s ", P);
                      storePaper(A,P).
+!storePaper(A,P): not databaseAvailable
                      <- .print("<<<Store Paper in System Local Disk
                         for ", A, "'s ", P, ">>>");
                          storePaperLocally(A,P).
+!sendAckToAuthor(A): internetAccess
                      <-  .print("Send Ack to ", A);
                          .send(A, tell, receiveAck).
```

```
+!sendAckToAuthor(A): not internetAccess
                    <- .print("<<<Send Ack to ", A, " via mail>>>");
                       .send(A, tell, receiveAck).
```

Listing 4.12: More reactive process management agent plan library(Reactivity 0.125)

```
/* Plans */
+receiveCameraReady[source(A)] <- .print("Camera-ready Received
    from ", A);
                                  cameraReadyReceived;
                                  .print("Collect Camera-ready");
                                  collectCameraReady;
                                  .print("Send to Publisher");
                                  sendToPublisher.
```

Listing 4.13: More reactive review management agent plan library(Reactivity 1.7667)

```
/* Plans */
+recevicereviewerDetail[source(A)] <- .print("<<<Review Detail
    Recived From ", A, ">>>");
                                      reviewerDetailRecevied;
                                      ?paperNum(N);
                                      .print("Invite Reviewers");
                                      inviteReviewers(N).
+accept[source(A)] <-   .print("Receive Accept to Review From ", A);
                         acceptToReview;
                        .print("Ask ",A," For Prefs ");
                        .send(A, tell, receivePref).
+reject[source(A)] <-   .print("Receive Reject to Review From ", A);
                        !dealWithRejection(A).
+!dealWithRejection(A): dealWithRejection
                       <- .print("<<<Deal With ", A, "'s Review
                          Reject>>>");
                       dealWithReject.
+!dealWithRejection(A): not dealWithRejection
                       <- rejectToReview.
+receivePrefs(R,N)[source(A)] <-  .print("Prefs Received From ",R,"
    On Author",N,"'s Paper");
                                  preferenceReceived;
```

```
                                        .print("Collect Prefs");
                                        collectPreference;
                                        .print("Assign Papers to ", R);
                                        .send(R, tell, receviePaper(N)).
+receiveReviewrReports(N)[source(A)] <- .print("<<<Review Reports
    Recivied On Paper",N," From ", A, ">>>");
                                             receiveReviewReport;
                                             .print("Collect Review
                                                Reports On Paper", N);
                                             collectReviewReport.
+receiveDecision[source(A)] <- .print("Decision Received");
                                decisionReceived;
                                .print("Finalise decision");
                                finaliseDecision;
                                .print("Send Notification");
                                ?paperNum(N);
                                .print("Send Notification");
                                sendNotification(N).
+emergency(N)[source(A)] <- .print("Receive Emergency
    Notification");
                            .print("<<<Invite Reviewer for Dealing
                                With Emergency on Paper",N, ">>>");
                            inviteReviewersInEmergency(N).
```

Listing 4.14: More reactive reviewer plan library(Reactivity 1.7667)

```
/* Initial goals */
+invitationReceived[source(A)]<-.print("Invitation Received");
                                   invitationReceived;
                                   !sendReply.
+!sendReply: accept
            <-  .print("Send Accept to review");
                .send(reviewManagement,tell, accept);
                -accpet.
+!sendReply :not accept
            <- .print("Send Reject to review");
               .send(reviewManagement, tell, reject).
+receivePref[source(R)] <-  .print("Prefs request received From ",
    R);
                            prefsRequstReceived;
```

```
                              .print("Send Prefs");
                               sendReviewPref.
+receviePaper(N)[source(R)] <- .print("Paper ",N," Received");
                                 paperReceived;
                                 !reviewPaper(N);
                                 !sendReviewReports(N).
+!reviewPaper(N):emergency
                    <- .print("<<<Send Emergency Notificationl>>>");
                       .send(reviewManagement, tell, emergency(N)).
+!reviewPaper(N): not emergency
                    <-.print("Review Paper", N);
                       reviewPaper.
+!sendReviewReports(N): not emergency
                         <-print("Send Review Reports on Paper", N);
                           .send(reviewManagement, tell,
                              receiveReviewrReports(N)).
+!sendReviewReports(N).
```

Listing 4.15: Trace of System Execution

```
[author1] Submit Paper
[author3] Submit Paper
[author4] Submit Paper
[author2] Submit Paper
[submissionManagement] Paper Received From author1
[submissionManagement] Paper Received From author3
[submissionManagement] Paper Received From author4
[submissionManagement] Paper Received From author2
[submissionManagement] Generate Paper ID For author2's Paper
[submissionManagement] Generate Paper ID For author1's Paper
[submissionManagement] Generate Paper ID For author4's Paper
[submissionManagement] Generate Paper ID For author3's Paper
[submissionManagement] Store Paper For author2's Paper
[submissionManagement] Store Paper For author1's Paper
[submissionManagement] Store Paper For author4's Paper
[submissionManagement] Store Paper For author3's Paper
[submissionManagement] <<<Send Ack to author2 via mail>>>
[author2] Ack received
[submissionManagement] <<<Send Ack to author1 via mail>>>
[submissionManagement] <<<Send Ack to author4 via mail>>>
```

```
[submissionManagement] <<<Send Ack to author3 via mail>>>
[author1] Ack received
[author4] Ack received
[author3] Ack received
[pcchair] Enter Reivewer Detail
[reviewManagement] Review Detail Recived From pcchair
[reviewManagement] Invite Reviewers
[reviewer1] Invitation Received
[reviewer4] Invitation Received
[reviewer8] Invitation Received
[reviewer4] Send Accept to review
[reviewer8] Send Accept to review
[reviewer1] Send Reject to review
[reviewer5] Invitation Received
[reviewManagement] Receive Accept to Review From reviewer4
[reviewer5] Send Accept to review
[reviewManagement] Ask reviewer4 For Prefs
[reviewManagement] Receive Accept to Review From reviewer8
[reviewManagement] Receive Reject to Review From reviewer1
[reviewManagement] Receive Accept to Review From reviewer5
[reviewer4] Prefs request received From reviewManagement
[reviewManagement] Ask reviewer8 For Prefs
[reviewer4] Send Prefs
[reviewManagement] <<<Deal With reviewer1's Review Reject>>>
[reviewManagement] Ask reviewer5 For Prefs
[reviewer8] Prefs request received From reviewManagement
[reviewer8] Send Prefs
[reviewManagement] Prefs Received From reviewer4 On Author1's Paper
[reviewManagement] Prefs Received From reviewer8 On Author2's Paper
[reviewer5] Prefs request received From reviewManagement
[reviewManagement] Collect Prefs
[reviewer5] Send Prefs
[reviewManagement] Collect Prefs
[reviewManagement] Prefs Received From reviewer5 On Author3's Paper
[reviewManagement] Assign Papers to reviewer4
[reviewManagement] Assign Papers to reviewer8
[reviewManagement] Collect Prefs
[reviewer4] Paper 1 Received
[reviewManagement] Assign Papers to reviewer5
[reviewer8] Paper 2 Received
```

```
[reviewer5] Paper 3 Received
[reviewer8] Review Paper2
[reviewer4] Review Paper1
[reviewer5] Review Paper3
[reviewManagement] <<<Review Reports Recivied On Paper2 From reviewer8>>>
[reviewManagement] <<<Review Reports Recivied On Paper1 From reviewer4>>>
[reviewManagement] Collect Review Reports On Paper2
[reviewManagement] <<<Review Reports Recivied On Paper3 From reviewer5>>>
[reviewManagement] Collect Review Reports On Paper1
[reviewManagement] Collect Review Reports On Paper3
[reviewer10] Invitation Received
[reviewer10] Send Accept to review
[reviewManagement] Receive Accept to Review From reviewer10
[reviewManagement] Ask reviewer10 For Prefs
[reviewer10] Prefs request received From reviewManagement
[reviewer10] Send Prefs
[reviewManagement] Prefs Received From reviewer10 On Author4's Paper
[reviewManagement] Collect Prefs
[reviewManagement] Assign Papers to reviewer10
[reviewer10] Paper 4 Received
[reviewer10] <<<Send Emergency Notification>>>
[reviewManagement] Receive Emergency Notification
[reviewManagement]<<<Invite Reviewer for Dealing With Emergency on Paper4>>>
[reviewer9] Invitation Received
[reviewer9] Send Accept to review
[reviewManagement] Receive Accept to Review From reviewer9
[reviewManagement] Ask reviewer9 For Prefs
[reviewer9] Prefs request received From reviewManagement
[reviewer9] Send Prefs
[reviewManagement] Prefs Received From reviewer9 On Author4's Paper
[reviewManagement] Collect Prefs
[reviewManagement] Assign Papers to reviewer9
[reviewer9] Paper 4 Received
[reviewer9] <<<Send Emergency Notification>>>
[reviewManagement] Receive Emergency Notification
```

```
[reviewManagement] <<<Invite Reviewer for Dealing With Emergency on Paper4>>>
[reviewer3] Invitation Received
[reviewer3] Send Accept to review
[reviewManagement] Receive Accept to Review From reviewer3
[reviewManagement] Ask reviewer3 For Prefs
[reviewer3] Prefs request received From reviewManagement
[reviewer3] Send Prefs
[reviewManagement] Prefs Received From reviewer3 On Author4's Paper
[reviewManagement] Collect Prefs
[reviewManagement] Assign Papers to reviewer3
[reviewer3] Paper 4 Received
[reviewer3] <<<Send Emergency Notification>>>
[reviewManagement] Receive Emergency Notification
[reviewManagement] <<<Invite Reviewer for Dealing With Emergency on Paper4>>>
[reviewer6] Invitation Received
[reviewer6] Send Accept to review
[reviewManagement] Receive Accept to Review From reviewer6
[reviewManagement] Ask reviewer6 For Prefs
[reviewer6] Prefs request received From reviewManagement
[reviewer6] Send Prefs
[reviewManagement] Prefs Received From reviewer6 On Author4's Paper
[reviewManagement] Collect Prefs
[reviewManagement] Assign Papers to reviewer6
[reviewer6] Paper 4 Received
[reviewer6] Review Paper4
[reviewManagement] <<<Review Reports Recivied On Paper4 From reviewer6>>>
[reviewManagement] Collect Review Reports On Paper4
[pcchair] Read Review
[pcchair] Make a Decision
[reviewManagement] Decision Received
[reviewManagement] Finalise decision
[reviewManagement] Send Notification
[reviewManagement] Send Notification
[author1] <<<Wait to be available>>>
[author3] Receive Reject
[author4] Receive Accept
[author2] Receive Accept
```

```
[author4] Prepare Camera-ready
[author2] Prepare Camera-ready
[author4] Send Camera-read
[author2] Send Camera-read
[processingManagement] Camera-ready Received from author4
[processingManagement] Camera-ready Received from author2
[processingManagement] Collect Camera-ready
[processingManagement] Collect Camera-ready
[processingManagement] Send to Publisher
[processingManagement] Send to Publisher
[author1] Receive Accept
[author1] Prepare Camera-ready
[author1] Send Camera-read
[processingManagement] Camera-ready Received from author1
[processingManagement] Collect Camera-ready
[processingManagement] Send to Publisher
```

By comparing execution results in 4.8 and 4.15, under the same environment, the second MAS for confluence submission process simulation has more alternative actions than the first one, marked as $<<< *** >>>$ in 4.15. That means the second agent system is more reactive than the first one. Because it can get more perceptions from the environment and react to the environment. For example, when the Internet is not available, the system can send acknowledgement letter to author via mail; when one reviewer has an emergency, the system can receive emergency notification and invite another reviewer. It can bee seen in library headers, reactivity for author plan library is increased from 0.9583 to 1.25; submission management agent plan library is increased from 0.1667 to 0.5833; review management agent plan library is increased from 0.9333 to 1.7667.

Based on the analysis above, a conclusion can be made that the measure on agent reactivity can be used to compare different agent plan libraries. The reactivity measure proposed in this thesis not only tells the increase of reactivity, but also illustrates how much it is increased. The only problem for the reactive measure is that it is not easy to calculate by hand, especially for some large MAS systems. Therefore, a tool such as the eclipse plugin mentioned in this thesis is necessary in calculating the reactivity measure. That tool should be robust enough to handle various syntaxes and formats in agent plan libraries.

Chapter 5

Conclusions and Future Work

Researchers and business communities have been interested in intelligent agent technology since 1980s. Agent's capability and features have been evidenced in various areas [39, 43, 17]. Agents have four key properties, namely autonomy, reactivity, proactiveness and sociability[53]. Although a lot of agent-oriented architectures and agent programming languages have been designed and proposed, there has been limited research on software quality measures specifically for agent-based systems. In this thesis, work towards a measure suite for agent systems, focusing specially for the famous BDI agent programs, has been described. This thesis also puts forward defintion on agent behavioural profile and behavioural preservation which can be used to check whether agent's original behaviours are preserved after reactivity enhancement.

One of the most significant properties of agents is they can make immediate response to changes in the environment they operate (i.e. reactivity). Several factors can affect the reactivity of an agent systems are argued in this thesis including: (1) the availability of plans to handle all external events relevant to the agents; and (2) the ability to delay their commitment to a certain course of action as late as possible (i.e. wait until the agent has the most updated knowledge about the environment) by preferring subgoals over primitive actions. Reactivity measures for agent programs that takes into these factors are proposed. An algorithm has also been proposed based on this measure to examine the plan library of an agent program and returns the reactivity of the program. A reactivity plug-in integrated with the Eclipse-based development environment of Jason, a well-known agent-oriented programming platform, has been implemented. In order to guarantee the reliability of reactivity measures, several experiments have been conducted.

When working on the agent reactivity, it is found that being too reactive to the environment is not a good thing in some cases. Some actions might depend on the state of the environment. For instance, only when it is raining, will people open umbrella. Therefore, before opening an umbrella, people may need to know whether it is raining or not. If it is not raining, opening an umbrella turns out to be an unnecessary action. In order to avoid inaccurate and unnecessary actions in goal-oriented behaviors, a state transition system for the environment can be developed. The state transition system is made up of states of the environment with available actions associated with each state. If an action is defined in a state, a checking on the state of the environment needs to be executed before performing the action. A model of the state transition system for goal-oriented reactive agent is shown as follows:

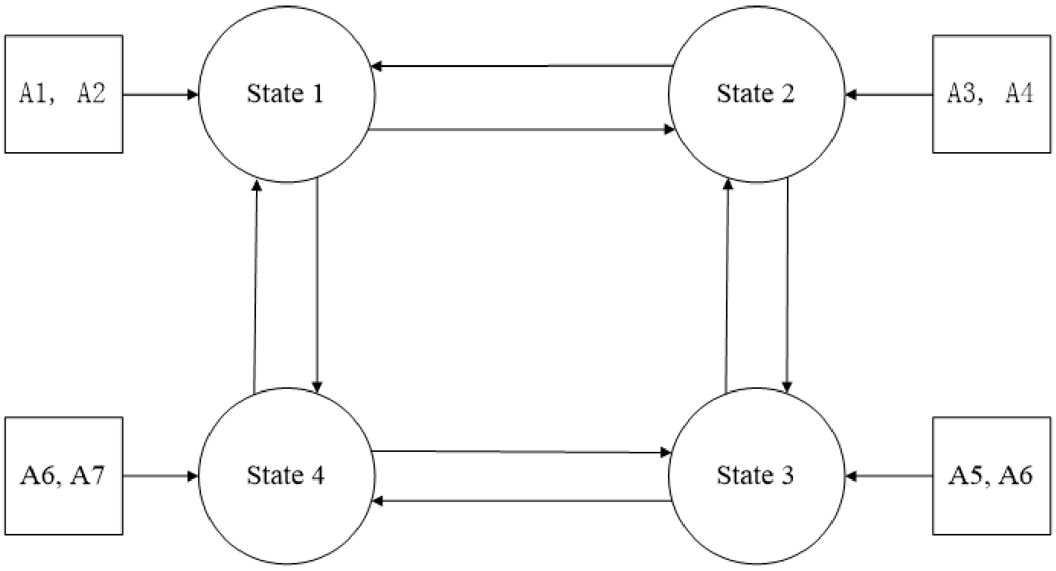


Figure 5.1: State Transition System for the Environment

In the model, there exists four states combined with actions that can be performed in each state. A1 and A2 can be performed in state 1; A3 and A4 can be performed in state 2; A5 and A6 can be performed in state 3 and A6 and A7 can be performed in state 4. For instance, if an agent wants to perform A3, a checking on the state of the environment needs to be performed. If current state is state 2, performing A3 proves to be a correct action. Otherwise, the agent might not reactive to the environment enough. Suppose a person wants to go to city by train. The plan can be expressed as: $+!to\_city : train_working \leftarrow go\_to\_station; wait\_for\_train; get\_on\_train; arrive$. However, it is possible that the train stops working when he is going to station or waiting for the train. Under such conditions, agent cannot make correct response if this plan is used. To tuning the plan, a state transition model for train can be set up. Each state contains its allowable actions.

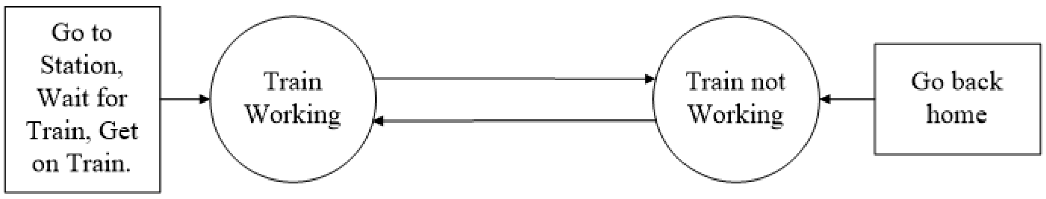


Figure 5.2: State Transition System for the Environment Example

According the state transition diagram, context condition checking is required before performing $go\_to\_station$, $wait\_for\_train$, get_on_train and go_back_home. Without context condition, the reactivity of agent to the environment would be reduced. By moving actions to condition-decorated sub-goals, the agent should be reactive enough now. Unlike the old plan library, the redesigned plan library allows the person go back home before waiting for the train or getting on the train if the train stop working.
$+!to\_city : train\_work \leftarrow go\_to\_station; !sg1; !sg2; arrive.$
$+!sg1 : train\_work \leftarrow wait\_for\_train.$
$+!sg1 : train\_not\_working \leftarrow go\_back\_home.$
$+!sg2 : train\_work \leftarrow get\_on\_train.$
$+!sg2 : train\_not\_working \leftarrow go\_back\_home.$

With the assistance of state transition system of the environment, agent will not be too reactive to the environment. That means, actions not defined in state transition system do not rely on any states of the environment. Therefore, placing a context condition before those actions cannot bring positive effect. Context condition for those actions may even bring performance issue to an agent. A state transition system of the environment brings some differences in calculating agent reactiveness. According to the current reactivity measure, actions without context condition in front of them should be counted. However, actions without context condition in front of them and appeared in state transition system should be considered as a negative effect to agent reactivivity if state transititon system for the environment is introduced.

As a state transition system for the environment is a future work of this research, the concept mentioned above might lack of consideration.

References

[1] F. Alonso, J. L. Fuertes, L. Martínez, and H. Soza. Measuring the social ability of software agents. In *Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications*, pages 3–10, Washington, DC, USA, 2008. IEEE Computer Society. 13

[2] F. Alonso, J. L. Fuertes, L. Martinez, and H. Soza. Towards a set of measures for evaluating software agent autonomy. In *Proceedings of the 2009 Eighth Mexican International Conference on Artificial Intelligence*, MICAI '09, pages 73–78, Washington, DC, USA, 2009. IEEE Computer Society. 14

[3] F. Alonso, J. L. Fuertes, L. Martinez, and H. Soza. Measuring the proactivity of software agents. *International Conference on Software Engineering Advances*, 0:319–324, 2010. 1, 12

[4] M. Andersson and P. Vestergren. Object-oriented design quality metrics, 2004. 16

[5] G. Barnes and B. Swim. Inheriting software metrics. *Journal of Object-Oriented Programming*, 6(7):27–34, November - December 1993. 22

[6] F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors. *Methodologies and Software Engineering for Agent Systems*. Kluwer Academic Publishing (New York), 2004. 1

[7] R. Bordini, L. Braubach, M. Dastani, A. El Fallah Seghrouchni, J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming

languages and platforms for multi-agent systems. In *Informatica 30*, pages 33–44, 2006. 8

[8] R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, 2005. 1, 8

[9] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying multi-agent programs by model checking. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 12:239–256, 2006. 58

[10] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley, 2007. ISBN 0470029005. 5, 8

[11] B. Bounabat, R. Romadi, and S. Labhalla. Designing multi-agent reactive systems: A specification method based on reactive decisional agents. In H. Nakashima and C. Zhang, editors, *Approaches to Intelligence Agents*, volume 1733 of *Lecture Notes in Computer Science*, pages 775–775. Springer Berlin / Heidelberg, 1999. 10.1007/3-540-46693-2_15. 11

[12] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988. 5

[13] B. Burmeister, M. Arnold, F. Copaciu, and G. Rimassa. BDI-Agents for agile goal-oriented business processes. In Padgham, Parkes, Müller, and Parsons, editors, *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 37–44, Estoril, Portugal, May 2008. 1

[14] P. Busetta, N. Howden, R. Rönnquist, and A. Hodgson. Structuring BDI agents in functional clusters. In *Agent Theories, Architectures, and Languages (ATAL-99)*, pages 277–289. Springer-Verlag, 2000. LNCS 1757. 8

[15] L. Cernuzzi, G. Rossi, and L. Plata. On the evaluation of agent oriented modeling methods. In *In Proceedings of Agent Oriented Methodology Workshop*, pages 21–30, 2002. 11, 13

[16] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, June 1994. 18

[17] H. K. Dam and M. Winikoff. Towards a next-generation aose methodology. *Science of Computer Programming*, 78(6):684 – 694, 2013. ¡ce:title¿Special section: The Programming Languages track at the 26th ACM Symposium on Applied Computing (SAC 2011) & Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments¡/ce:title¿. 1, 85

[18] A. Dasgupta and A. K. Ghose. Implementing reactive BDI agents with user-given constraints and objectives. *Int. J. Agent-Oriented Softw. Eng.*, 4(2):141–154, Apr. 2010. 10

[19] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In *ATAL '97: Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages*, pages 155–176, London, UK, 1998. Springer-Verlag. 8

[20] J. Doyle. Rationality and its roles in reasoning. *Computational Intelligence*, 8:376–409, 1994. 5

[21] C. Gall, S. Lukins, L. Etzkorn, S. Gholston, P. Farrington, D. Utley, J. Fortune, and S. Virani. Semantic software metrics computed from natural language design specifications. *Software, IET*, 2(1):17 –26, february 2008. 18

[22] R. Gunnalan, M. Shereshevsky, and H. Ammar. Pseudo dynamic metrics [software metrics]. In *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on*, page 117, 2005. 18

[23] M. H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Ltd, 1977. 15

[24] W. A. Harrison and K. I. Magel. A complexity measure based on nesting level. *SIGPLAN Not.*, 16(3):63–74, Mar. 1981. 15, 16

[25] B. Henderson-Sellers and P. Giorgini, editors. *Agent-Oriented Methodologies*. Idea Group Publishing, 2005. 1

[26] K. V. Hindriks, F. S. D. Boer, W. V. D. Hoek, and J.-J. C. Meyer. Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999. 8

[27] T. Honglei, S. Wei, and Z. Yanan. The research on software metrics and software complexity metrics. In *Computer Science-Technology and Applications, 2009. IFCSTA '09. International Forum on*, volume 1, pages 131 –136, dec. 2009. 15

[28] M. J. Huber. Agent autonomy: Social integrity and social independence. In *Information Technology, 2007. ITNG '07. Fourth International Conference on*, pages 282 –290, april 2007. 13

[29] S. Husein and A. Oxley. A coupling and cohesion metrics suite for object-oriented software. In *Computer Technology and Development, 2009. ICCTD '09. International Conference on*, volume 1, pages 421 –425, nov. 2009. 17

[30] C. A. Iglesias, M. Garijo, J. C. González, and J. R. Velasco. Analysis and design of multiagent systems using MAS-commonKADS. In *Agent Theories, Architectures, and Languages*, pages 313–327, 1997. 12

[31] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert: Intelligent Systems and Their Applications*, 7(6):34–44, 1992. 8

[32] K. Kim, Y. Shin, and C. Wu. Complexity measures for object-oriented program based on the entropy. In *Software Engineering Conference, 1995. Proceedings., 1995 Asia Pacific*, pages 127 –136, dec 1995. 16

[33] D. Kinny, M. Georgeff, and A. Rao. A methodology and modelling technique for systems of BDI agents. In R. van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, 1996. 12

[34] W. Li. Software product metrics. *Potentials, IEEE*, 18(5):24 –27, 1999/jan 1999. 15

[35] Z. Lin and K. Carley. Proactive or reactive: An analysis of the effect of agent style on organizational decision-making performance. *Intelligent Systems in Accounting Finance and Management*, 2:271–287, 1993. 12

[36] C. E. Martin, K. S. Barber, and K. S. Barber. Agent autonomy: Specification, measurement, and dynamic adjustment. In *In Proceedings of the Autonomy Control Software Workshop, Agents '99*, pages 8–15, 1999. 13

[37] I. Mathieson, S. Dance, L. Padgham, M. Gorman, and M. Winikoff. An open meteorological alerting system: Issues and solutions. In V. Estivill-Castro, editor, *Proceedings of the 27th Australasian Computer Science Conference*, pages 351–358, Dunedin, New Zealand, 2004. 1, 20

[38] T. McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, SE-2(4):308 – 320, dec. 1976. 15

[39] S. Munroe, T. Miller, R. A. Belecheanu, M. Pěchouček, P. McBurney, and M. Luck. Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents. *Knowledge Engineering Review*, 21(4):345–392, 2006. 1, 85

[40] L. Padgham and M. Winikoff. *Developing intelligent agent systems: A practical guide*. John Wiley & Sons, Chichester, 2004. ISBN 0-470-86120-7. 1

[41] P. Piwowarski. A nesting level complexity measure. *SIGPLAN Not.*, 17(9):44–50, Sept. 1982. 16

[42] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI reasoning engine. In R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming*, pages 149–174. Springer Science+Business Media Inc., USA, 9 2005. Book chapter. 8

[43] M. Pěchouček and V. Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 17:397–431, 2008. 1, 85

[44] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. V. de Velde and J. Perrame, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, pages 42–55. Springer Verlag, 1996. LNAI, Volume 1038. 2

[45] A. S. Rao and M. P. Georgeff. BDI agents: From theory to practice. Technical Note 56, Australian Artificial Intelligence Institute, Apr. 1995. 5

[46] O. Simonin and F. Gechter. An environment-based methodology to design reactive multi-agent systems for problem solving. In D. Weyns, H. Van Dyke Parunak, and F. Michel, editors, *Environments for Multi-Agent Systems II*, volume 3830 of *Lecture Notes in Computer Science*, pages 32–49. Springer Berlin / Heidelberg, 2006. 10.1007/11678809_3. 11

[47] R. So and L. Sonenberg. Situation awareness in intelligent agents: foundations for a theory of proactive agent behavior. In *Intelligent Agent Technology, 2004. (IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*, pages 86 – 92, sept. 2004. 11, 12

[48] H. Tao and Y. Chen. A metric model for trustworthiness of softwares. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on*, volume 3, pages 69 –72, sept. 2009. 18

[49] J. Thangarajah, S. Sardina, and L. Padgham. Measuring plan coverage and overlap for agent reasoning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 1049–1056, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems. 14

[50] M. Weidlich, J. Mendling, and M. Waske. Computation of behavioural profiles of processs models. *Technical report 08-2009*, June 2009. 42

[51] M. Weidlich, M. Weske, and J. Mendling. Change propagation in process models using behavioural profiles. In *Services Computing, 2009. SCC '09. IEEE International Conference on*, pages 33–40, 2009. 19

[52] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons (Chichester, England), 2002. ISBN 0 47149691X. 1, 4

[53] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995. 2, 5, 85

[54] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995. 9

[55] M. Zheng and O. Ormandjieva. Reliability analysis in the early development of real-time reactive systems. In *Computer Science and Information Engineering, 2009 WRI World Congress on*, volume 7, pages 807 –812, 31 2009-april 2 2009. 14

[56] K. Q. Zhu, W.-Y. Tan, A. E. Santosa, and R. H. C. Yap. Reactive web agents with open constraint programming. In *Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems*, ISADS '01, pages 251–, Washington, DC, USA, 2001. IEEE Computer Society. 10