A general framework for parallel BDI agents in dynamic environments

Huang, Shell Ying; Zhang, Huiliang

2006

Zhang, H., & Huang, S. Y. (2006). A general framework for parallel BDI agents in dynamic environments. IEEE/WIC/ACM International Conference on Intelligent Agent Technology. IAT 2006, pp.1-23.

<https://hdl.handle.net/10356/90707>

<https://doi.org/10.1109/IAT.2006.8>

© 2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE. This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. <http://www.ieee.org/portal/site> This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

A general framework for parallel BDI agents in dynamic environments

Huiliang Zhang and Shell Ying Huang

School of Computer Engineering, Nanyang Technological University, Singapore 639798

E-mail: zhlpku@yahoo.com, ASSYHUANG@ntu.edu.sg

Abstract. In this paper, a general framework for the parallel BDI model suitable for dynamic environments is proposed. It is a parallel agent architecture that supports the following agent abilities at architecture level: (1) the ability to monitor the environment at all times and respond to emergencies timely; (2) the ability to reconsider and re-schedule goals, intentions and actions in reaction to unexpected or new information; (3) the ability to perform multiple actions at once; (4) the ability to perceive, deliberate and act simultaneously; (5) the ability to prioritize the deliberations and intention executions. We define the functions and the operations of the processing units in the agent and how these units interact, cooperate and synchronize with each other. With the advances in semiconductor technology which allow multiple processing units to be implemented on the same silicon chip, a parallel BDI agent will be an effective way to enable it to perform in a dynamically changing environment when the arrival rate of events is high. We illustrate the working of a parallel agent under the general framework with an agent simulating the behaviour of a vessel captain navigating in sea. Then the performance of a parallel agent is evaluated against several versions of sequential agents. The issue of how much parallelism and how to configure a parallel agent based on the general framework are studied by experiments with different configurations of the parallel agent.

Keywords: Agent architecture, parallel activities.

1 Introduction

Hayes-Roth [18] defined the primary objective of an intelligent agent that needs to perform in real time as “to maintain the value of its own behaviour within an acceptable range over time”. Among the requirements for an intelligent agent, two related are flexibility (the agent should react to important unexpected events) and timeliness (the agent should meet various real-time constraints). Many agent architectures or frameworks have been developed for building an intelligent agent. As identified in the survey of agent architectures [37], three kinds of agent architectures, deliberative architecture [5, 32, 20, 31, 36, 35], reactive architecture [7, 8] and hybrid architecture [14, 15, 25], are proposed according to the processing mechanism of the agents. The BDI (belief-desire-intention) model is well understood for designing deliberative architectures because it combines a respectable philosophical model of human practical reasoning [16]. The mental attitudes of belief, desire and intention represent the information, motivational, and deliberative states of the agent

respectively [5, 32]. The reactive architecture, most noticeably the subsumption architecture, is a different paradigm [6, 7, 8]. A reactive system generally provides quicker response than the deliberative architecture but is hard to create and maintain [39]. The hybrid architecture is proposed to combine the deliberative and reactive architectures to inherit the advantages of both.

With the different agent models proposed, it is said [16] that the basic components of an agent designed for a dynamic, uncertain world should include some representation of Beliefs, Desires, Intentions and Plans – the BDI model. There are three main operations in this model: detecting, thinking and executing. In the normal implementation of deliberative agents, the three operations run sequentially. For example, in PRS (procedural reasoning system) [20], the deliberative process runs sequentially in iterations. At the beginning of each iteration step, new goals and new facts are obtained through input. Then several plans in the KA (knowledge area) library are triggered by the new belief and one or more of the applicable plans are selected to be sent to the intention structure. At the end of each

iteration step, the intentions are executed. This idea of implementing the BDI agent is adopted in many BDI systems [30, 1, 19, 10]. In PRS, the agent will not proceed to the next step until the current step is finished. In a complex and dynamic environment, the agent needs more time to search for proper intentions or one action may need more time to execute. Then more time is needed in each iteration step and the agent is not able to detect new events before the current iteration step is finished. As a consequence of this, the agent may not be able to start processing the emergencies immediately. So the reactivity of the PRS agent cannot be assured in such circumstances.

A probable solution to promote the reactivity of the agents appeared in the TouringMachine [14], a hybrid agent architecture. The architecture consists of three sequential components: perception subsystem, control framework and action subsystem. The control framework will output actions to the action subsystem based on the sensory input from the perception subsystem. The control framework consists of three layers, a reactive layer, a planning layer and a modeling layer. The outputs from the three layers are summarized by using some context-activated control rules. A clock is used to control the execution time of the control framework. For each cycle, the control framework has fixed time resource to use. A primitive schema (action) structure is designed with a 'cost' property, which indicates how much time it costs to execute an action. If in a cycle, the remaining time resource is not sufficient for an action to execute, another action with less time cost will be executed instead or the remaining time is wasted if no suitable action is available. This method ensures that the agent can sense the environment at fixed time intervals. The probability of overlooking emergencies is low if the time spent in detection and processing is balanced well. But it is usually advised that the detection should not consume much time. So if the TouringMachine puts much time on the control framework in a cycle, the problem of poor reactivity still exists.

In agents based on BDI logics, such as AgentSpeak(L) [33] and LORA (Logic of Rational Agents) [38], there are issues such as reactivity and intention reconsideration. Reasoning in these agents is done by theorem provers, which usually need much executing time. In LORA, the basic agent control loop of the BDI interpreter consists of perception, updating belief, generating desires, choosing intention and executing actions. Desires, intentions, and actions are generated based on beliefs. The original circumstance/belief may have changed during these processing. Some intentions may become impossible under the new circumstance. The agent should not continue with its commitment to the infeasible

intentions. An improvement is made by updating belief and reconsidering intentions after executing each action. An experimental result of intention reconsideration by Kinny and Georgeff is provided in the book [38]. The result shows that the more frequently the intentions are reconsidered, the lower the effectiveness of the agent is. Thus, the reactivity of the agent cannot be ensured. This shows that it is important for the agent to know when an intention should be reconsidered and not to do it unnecessarily.

Extensions and improvements to the original BDI reasoning cycles are done in [10, 4, 9]. In Jason [4], an agent can use customized selection functions to choose the events/intention plans to process/execute in each cycle. However, the agent still goes through the same perception and deliberation steps in each iteration. The significance of the work in 3APL is that the deliberation process is made programmable [9, 10]. One cycle of the deliberation process consists of the goal revision phase and the goal execution phase. In the goal revision phase, practical reasoning rules can be applied to dynamically generate new goals and revise existing goals, plan and replan. The steps in the goal revision phase and goal execution phase roughly correspond to those in a BDI reasoning cycle. Selected reasoning rules may be applied to selected goals to make an agent more reactive or deliberative. When there are no rules to apply, control will be passed quickly from the goal revision phase to the goal execution phase. This work is closest to what we want to achieve in a parallel BDI agent but the cost in executing the practical reasoning rules in 3APL will make a 3APL agent less reactive when it needs to be.

Pokahr [28] suggested that in the sequential BDI agents the concrete layout of the processing cycle will determine the nature of the agent, for example, the caution level and reconsideration rate. The agent architecture is also not easy to be extended with additional facilities because the processing is step by step and very restrictive. The authors proposed a more flexible way of mapping the BDI model to allow easy extension of the agents. In the architecture, the steps are transformed to meta-actions. A main interpreter will decide which meta-action will be selected to execute from the agenda queue. The execution of the meta-action may update the status and insert new meta-actions to agenda. The extension of new agent abilities can be easily done by designing new types of meta-actions. However, it is noticed that the outside messages are inserted into the agenda directly as external actions. If perceiving environment is modeled as a regular meta-action, there is no guarantee at the architecture level that the environment is monitored appropriately closely. At the same time, the problems of low caution level and reconsideration rate also remain.

Pokahr [29] commented that the current BDI model does not support any mechanism for handling goal relationships at the architecture level. They proposed a deliberation strategy for agent developers to specify relationships between goals such that there are a maximum number of goals that an agent may pursue at once and the activation of one goal may inhibit another goal. However, another important factor that is not considered is the importance and urgency of a goal that influences which goal should have the attention of the agent first.

Hayes-Roth [18] pointed out that parallel subsystems with buffered communications to provide asynchronous perception, cognition and action will allow an agent to perform in real time. We propose what are required of an agent for real time performance: (1) ability to monitor the environment at all times and respond to emergencies timely; (2) ability to reconsider and re-schedule goals, intentions and actions in reaction to unexpected or new information; (3) ability to perform multiple actions at once (e.g. talking while walking); (4) ability to perceive, deliberate and act simultaneously (e.g. thinking while walking). Reconsideration of goals, intentions and actions for possible modification is not done before the execution of every deliberation, intention and action to avoid doing it excessively. It is done after the generation of a new belief, a new desire and a new intention by examining what is affected by a new belief, a new desire and a new intention.

In [22], a multi-threaded approach is used to simulate soccer agents for the RoboCup competition. The sensing, thinking and acting behaviours are executed in parallel. Thus the soccer agent does not need to wait for IO operations (sensor and act) with the world and gains more time for thinking. The experiments show that the agent with a parallel architecture has obvious advantages in lessening the impact of IO operations in the simulation of an intelligent agent like a human being.

In this paper, we propose a general framework for real time performance in the BDI model. The general idea is that such a framework consists of three main components, the belief manager, the intention generator and the intention executor as shown in Fig. 1. The three components each consists of a number of smaller processing units and they run in parallel. The coordination between them is done by messages and interrupts of different priorities. The belief manager generates beliefs from world information perceived by the agent and from interactions with human or other agents. The intention generator generates desires, then schedules and reschedules the generation of intention plans for the desires. The intention executor schedules and reschedules the execution of intention plans and executes them. Under this general framework, parallel

Fig. 1. The general idea of a parallel BDI agent model.

BDI agents with different configurations based on the best way to utilize the available computational resources may be built. These agents have a number of advantages over the sequential one: 1. they have the 4 abilities required of an agent as discussed earlier; 2. the support is provided at the architecture level for reconsideration of desires and intentions and the consideration of goal relationships when a new belief/desire is generated. The work presented in this paper is a revised and extended version of our work presented in [40, 42, 43].

The idea of parallel agents has appeared in the subsumption systems [6, 21], in which the layers are run concurrently. In LSA (Logic-Based Subsumption Architecture), the layers are realized as theorem provers. So the reasoning ability of the deliberative agents is combined into the reactive agents. The empirical results from a robot implementation show that the provers can be used without sacrificing much reactivity [2, 3]. This kind of parallel deliberative architecture is different from ours. In LSA, the layers are presumed to be independent. Each parallel layer will perform the activities of detection, reasoning and output sequentially. It can be seen as several deliberative agents each with its own sub-goals running in parallel in this subsumption architecture.

The remainder of this paper is structured as follows. In section 2, we present the general framework for parallel BDI agents that need to perform in real time in a dynamic environment. The functions of the processing units in the framework are identified, their operations are defined and how these functional units interact and cooperate is specified. Section 3 illustrates the working of a parallel agent under the general framework with an agent simulating the behaviour of a vessel captain navigating in sea. It demonstrates the functional capability of the parallel agent. In section 4,

Fig. 2. The General Framework for Parallel BDI Agents.

simulation experiments are presented to compare the performance of the parallel agent and five versions of sequential agents. In section 5, the issue of how much parallelism and how to configure a parallel agent based on the general framework with a limited number of processing units are studied by experiments with different configurations of the parallel agent. At the end of the paper, we summarize the contribution of the framework in Section 6.

2 The General Framework

We propose a general framework for a parallel deliberation system based on the BDI model. As shown in Fig. 2, the framework consists of three main components: the belief manager, the intention generator and the intention executor. These three components represent the three steps in the deliberation process of an agent: *detect*, *think* and *act* respectively. The three components will retrieve and update data in the three data structures: *beliefs*, *desires*, and *intentions*. The *beliefs* contain the agent's view of its environment and of itself. The *desires* are the goals to be achieved or states to be reached by the agent. The *intentions* store the action plans to be executed to achieve the agent's goals. Each of the three components are further divided into smaller processing elements, also shown in Fig. 2. We define the functionalities of each processing element and the interactions among them.

2.1 The Belief Manager

The belief manager is responsible for detecting the changes in its environment and managing the agent's view of the world and of itself. It is made up with Environment Monitors (EMs) and a Belief Generator (BG). The set of EMs serve as the information collectors from heterogeneous sensors that an agent may have. Each EM monitors world information from one type of sensors or sensory organs like a camera or human eyes and converts the information into abstract representation. Each EM sends the converted information to the BG.

The BG will merge the various information items passed from the EMs into the agent's view of the world. For example, the eyes of a person see and the ears hear a car coming. The visual and audio information will come through two separate EMs and the BG combines the information to form a new belief. This is a cognitive process where the semantics of the sensor information is worked out. Each new belief has a certain degree of urgency. The BG will determine the urgency of the new belief. When a new belief is generated, existing contradictory or obsolete beliefs will be removed. In other words, when b_{new} , a new belief is formed, we have

$$beliefs = beliefs \cup \{b_{new}\} \text{ and } \exists b \in beliefs[obsolete(b_{new}, b)] \rightarrow beliefs = beliefs - \{b\}$$

Messages will be produced by the BG to notify the intention generator. Depending on the urgencies of the

new beliefs, message of different priority levels will be generated so that the intention generator may process them accordingly. This allows urgent information to be handled immediately and less urgent information be dealt with later. As the belief manager works simultaneously with the intention generator and the intention executor, this allows the agent to monitor its environment including any emergencies at all times.

2.2 The Intention Generator

The intention generator manages the agent's desires(goals), based on the agent's current beliefs. This includes generation of new goals, removal of obsolete goals and suspension of existing goals. The intention Generator also deliberates on the plans to achieve these goals. It is made up with a Desire Generator (DG), a Desire Scheduler (DS) and a number of Plan Generators (PGs).

The DG may produce new desires after a new belief is added into *beliefs*. The new desire will have a priority level according to these rules:

If $b_1 \wedge b_2 \wedge \dots \wedge b_n \Rightarrow d$ then $pd = \max(pb_i)$, $i = 1 \dots n$.

If $b_1 \vee b_2 \vee \dots \vee b_n \Rightarrow d$ then $pd = \max(pb_i)$,
 $i = 1 \dots n$, and $b_i \in \text{beliefs}$

These rules mean that when a number of beliefs b_1 to b_n conjunctively trigger a desire d , the priority of the desire pd , will have the highest priority level of the beliefs that triggered its generation. When a desire d may be triggered by any one of a number of beliefs $b_1 \dots b_n$, the priority of the desire will have the highest priority level of the belief among $b_1 \dots b_n$ that the agent currently believes in.

A new belief may make a current desire no longer desirable because it is obsolete or it is not consistent with the new desire. A new belief may also affect the priority of an existing desire. After the generation of a new desire, the new desire may affect an existing desire or be affected by an existing desire with respect to their priorities. The intention generator will handle these cases as described later.

The set of desires of an agent *desires* consists of three subsets:

$$\text{desires} = \text{pendingD} \cup \text{planningD} \cup \text{executingD}$$

where

pendingD: the set of desires waiting for their plans to be worked out or a partial plan for the desire has been decided but more deliberation is needed,

planningD: the set of desires being planned, that is, a plan to achieve the desire is being worked out in a PG,

Fig. 3. States of desires and intentions and their transition.

executingD: the set of desires whose intention plans are waiting to be executed or being executed by the intention executor.

If $d_i \in \text{desires}$ then $d_i = (ID_i, g_i, p_i, s_i)$ where ID_i is the identifier of the desire, g_i is the desired goal of the agent, p_i is the priority of g_i representing its urgency and s_i represents the status of deliberation about this goal.

With the partition of *desires* into *pendingD*, *planningD* and *executingD* as shown in Fig. 3, we provide a simple way to handle the reconsideration of goals. When a desire is generated by the DG, it is deposited into the *pendingD*. A desire will move from *pendingD* to *planningD* when the Desire Scheduler (DS) decides that this desire should preempt another desire in *planningD* or there is a free PG to do planning and this desire has the highest priority in *pendingD*. A desire will be moved from *planningD* to *pendingD* by DS when this desire is preempted by another desire in *pendingD* or a new desire in *pendingD* calls for the suspension of this desire in *planningD*. A desire moves from *planningD* to *executingD* when the intention plan for the desire has been generated. A desire will be removed from *pendingD*, *planningD* or *executingD* if a new belief or a new desire makes the agent abandon this desire.

We define the operations of DG in Fig. 4 and those for DS in Fig. 5. In Fig. 4, $\text{obsolete}(a, d)$ means belief/desire a makes desire d obsolete. $\text{urgencyAffected}(a, d)$ means belief/desire a changes the urgency of desire d or desire d changes the urgency of desire a . This may result in an increase or a decrease of priority for d or a (if a is a desire), or a temporary suspension of d or a . Suspension of a desire happens when an agent decides to shelf the desire temporarily because of the conflicts between a new desire and an existing desire. $\text{clash}(d_1, d_2)$ in Fig. 5 returns true if d_2

was suspended (priority set to suspensionPriority) because of d_i . The function $\text{preempt}(p_k, p_i)$ in Fig. 5 will decide whether d_k should preempt d_i . For example, suppose d_k is the desire with the highest priority in pendingD , d_i is the desire with the lowest priority in planningD and $\text{preempt}(p_k, p_i) = p_k > p_i$. Then at all times, we have

$$\forall i \forall j, (ID_i, g_i, p_i, s_i) \in \text{pendingD}, (ID_j, g_j, p_j, s_j) \in \text{planningD} [p_i \leq p_j]$$

It is possible to have other more elaborate definitions for $\text{preempt}(p_k, p_i)$, for example, $p_k > (p_i - \text{some threshold})$ and $(\text{remaining time for } d_i > \text{some threshold})$.

LOOP:

b_{new} = the belief with maximum priority among the newly generated beliefs in *beliefs*;
 Process the new belief b_{new} and form a new desire d_{new} , if appropriate;
 If $d_{\text{old}} \in \text{desires}$ and $(\text{obsolete}(b_{\text{new}}, d_{\text{old}}))$
 If $d_{\text{old}} \in \text{pendingD}$
 $\text{pendingD} = \text{pendingD} - \{d_{\text{old}}\}$.
 If $d_{\text{old}} \in \text{planningD}$
 Set the priority of d_{old} to removalPriority such that DS will remove it from a PG.
 If $d_{\text{old}} \in \text{executingD}$
 Notify IM through interrupts to have the corresponding intention removed.
 If $d_{\text{old}} \in \text{desires}$ and $(\text{urgencyAffected}(b_{\text{new}}, d_{\text{old}}))$
 If $d_{\text{old}} \in \text{pendingD} \cup \text{planningD}$
 Change the priority of d_{old} to a higher/lower value or suspensionPriority.
 If $d_{\text{old}} \in \text{executingD}$
 Notify IM through interrupts to change the priority of the corresponding intention.
 If a new desire, d_{new} , is formed
 $\text{pendingD} = \text{pendingD} \cup \{d_{\text{new}}\}$.
 If $d_{\text{old}} \in \text{desires}$ and $(\text{obsolete}(d_{\text{new}}, d_{\text{old}}))$
 If $d_{\text{old}} \in \text{pendingD}$
 $\text{pendingD} = \text{pendingD} - \{d_{\text{old}}\}$.
 If $d_{\text{old}} \in \text{planningD}$
 Set the priority of d_{old} to removalPriority such that DS will remove it from a PG.
 If $d_{\text{old}} \in \text{executingD}$
 Notify IM through interrupts to have the corresponding intention removed.
 If $d_{\text{old}} \in \text{desires}$ and $(\text{urgencyAffected}(d_{\text{new}}, d_{\text{old}}))$
 If $d_{\text{old}} \in \text{pendingD} \cup \text{planningD}$
 Change the priority of d_{old} or d_{new} to a higher/lower value or suspensionPriority.
 If $d_{\text{old}} \in \text{executingD}$
 Notify IM through interrupts to change the priority of the corresponding intention.
 If there is change in membership in pendingD or planningD
 Inform DS by message.

Fig. 4. Operations of DG.

LOOP:

1. **Scheduling desires according to the priorities of desires.**

LOOP1:

$$p_k = \max_{d_j \in \text{pendingD}} (p_j);$$

If p_k equals null or suspensionPriority
 break LOOP1.

If there is no free PG

$$p_i = \min_{d_j \in \text{planningD}} (p_j);$$

If $\text{preempt}(p_k, p_i)$

$$\begin{aligned} \text{pendingD} &= \text{pendingD} \cup \{d_i\}; \\ \text{planningD} &= \text{planningD} - \{d_i\}; \end{aligned}$$

```

        Send an interrupt to the PG to suspend  $d_i$ ;
    else
        break LOOP1.
    If there is a free PG
         $pendingD = pendingD - \{d_k\}$ ;
         $planningD = planningD \cup \{d_k\}$ ;
        Send an interrupt to the PG to start  $d_k$ .
2. Handling the completion of the planning of a desire by a PG.
    For each desire  $d_p \in planningD$ 
        If the planning of  $d_p$  is finished in a PG
             $planningD = planningD - \{d_p\}$ ;
             $executingD = executingD \cup \{d_p\}$ ;
            If  $d_{old} \in pendingD$  and  $clash(d_p, d_{old})$ 
                Increase the priority of  $d_{old}$  such that  $d_{old}$  may be considered for plan generation.
            Send a message to IM with the information about  $d_p$ .
3. Updating the priorities of desires.
    For each desire  $i \in desires$ 
        Update the priority of  $d_i$  according to DG instructions or by other factors, like time;
        If priority of  $d_i$  equals removalPriority
            If  $d_i \in executingD$ 
                Notify IM through interrupts to have the corresponding intention removed;
            else
                 $desires = desires - \{d_i\}$ ;
                If  $d_i \in planningD$ 
                    Send an interrupt to the PG to remove  $d_i$ .
        If priority of  $d_i$  equals suspensionPriority
            If  $d_i \in planningD$ 
                 $planningD = planningD - \{d_i\}$ ;
                 $pendingD = pendingD \cup \{d_i\}$ ;
                Send an interrupt to the PG to suspend  $d_i$ .
            If  $d_i \in executingD$ 
                Notify IM through interrupts to have the corresponding intention suspended.

```

Fig. 5. Operations of DS.

2.3 Intention Executor

The intention executor works in a similar way as the intention generator. Intention Manager (IM) will receive the plan information from the PGs in the intention generator. Intention Scheduler (IS) will schedule/suspend/resume the running of intentions in the Plan Executors (PEs). Finally, the physical actions will be put into the action effectors and the PE will remove the finished desire d from $executingD$. The agent's *intentions* consists of three subsets of intentions:

$$intentions = inactiveI \cup pendingI \cup executingI$$

where

inactiveI: the set of intentions that are waiting for the completion of other intentions before they can execute and intentions the agent has to put on hold temporarily,

pendingI: the set of intentions waiting to be executed,

executingI: the set of intentions being executed.

If $i_i \in intentions$ then $i_i = (ID_i, predecessors_i, peers_i, plan_i, p_i, s_i)$ where ID_i is the identifier, $plan_i$ is the action plan, p_i is the priority, s_i represents the status of the intention, $predecessors_i$ and $peers_i$ are used to address the dependency issues between different intentions and facilitate synchronization among intentions and is explained in the following paragraphs.

Normally, an intention plan may be a sequence of primitive actions to be carried out by the agent one after another. Or it may be a composite hierarchical task structure where predecessor subtasks need to be completed first and peer subtasks need to be executed simultaneously. If two subtasks are not related as predecessor-successor or as peer, they may be executed in any order or in parallel. Such a composite hierarchical task structure is transformed into several intentions by the PG which generates the intention plan in Intention Generator. The transformed intentions have a *predecessors* attribute to name the predecessor intentions which has to be completed before its own

Fig. 6. Transformation of a normal intention plan.

execution. They also have a *peer* attribute to identify what other intentions are to be executed simultaneously with them. An example of the transformation is shown in Fig. 6.

In the example in Fig 6, the original intention plan is transformed into 6 intentions with smaller action plans. Normally, each sequence of actions, like the one presented by Action a, is transformed into a new intention. If some sequences of actions are to be executed one by one, like Actions d and f, it may be possible to combine them into one intention like Intention 4, provided that both d and f may be executed by the same PE or the PEs are homogeneous. This reduces the cost for scheduling. In the rectangles showing the intentions 1-6, the first pair of bracelets shows the *predecessors* of the intention. For example, intention 6 cannot start before both intentions 4 and 5 are completed. With the transformation, the original intention plan can exploit parallelism supported in the agent's framework. An example is that intention 2 (Action b) and intention 3 (Action c) may be executed at the same time if two PEs are available. This speeds up the execution of intentions and the parallel framework of the agent is used more effectively.

Intentions 4 and 5 are peers so they have to be executed in 2 PEs simultaneously. The synchronization issue among peer intentions is discussed in the next section.

As shown in Fig. 3, *intentions* are partitioned into *inactiveI*, *pendingI* and *executingI*. This supports the scheduling and the reconsideration of intentions. An intention plan in *inactiveI* is one that can only start execution after the completion of its predecessor intentions or it is one that the agent wants to put on hold for the moment. An intention plan in *pendingI* or *executingI* has the same meaning as a desire in *pendingD* and *planningD* respectively. When a new intention plan arrives at IM, it joins *pendingI* if it has no predecessor intentions otherwise it joins the *inactiveI*. For each intention in *inactiveI*, IS checks whether all the predecessor intentions are completed. If yes, it moves this intention from *inactiveI* to *pendingI*. In this way, scheduling intentions with predecessors are easily managed by updating their *predecessor* attributes and organize them in *inactiveI* and *pendingI*.

The operations of IM and IS are defined in Fig. 7 and Fig. 8 respectively. Functions with the same names in Fig. 7 and Fig. 8 as those in Figs 4 and 5 behave in the same way.

LOOP:

1. **Handling the message, m_{new} , from DG.**

If $i_{old} \in intentions$ and ($obsolete(m_{new}, i_{old})$)

If $i_{old} \in inactiveI$

$executingD = executingD - \{d_{old}\};$

$inactiveI = inactiveI - \{i_{old}\}.$

If $i_{old} \in pendingI$

$executingD = executingD - \{d_{old}\};$

$pendingI = pendingI - \{i_{old}\}.$

If $i_{old} \in executingI$

Set the priority of i_{old} to *removalPriority* such that IS will remove it from a PE.

If $i_{old} \in intentions$ and ($urgenceAffected(m_{new}, i_{old})$)

Change the priority of i_{old} to a higher/lower value or *suspensionPriority*.

2. **Handling the message, m_{new} , from DS.**

```

Form a new intention,  $i_{new}$ , based on  $m_{new}$ ;
If  $i_{new}$  and its peers intentions have no predecessors intentions
     $pendingI = pendingI \cup \{i_{new}\}$ ;
else
     $inactiveI = inactiveI \cup \{i_{new}\}$ .
If  $i_{old} \in intentions$  and ( $obsolete(i_{new}, i_{old})$ )
    If  $i_{old} \in inactiveI$ 
         $executingD = executingD - \{d_{old}\}$ ;
         $inactiveI = inactiveI - \{i_{old}\}$ .
    If  $i_{old} \in pendingI$ 
         $executingD = executingD - \{d_{old}\}$ ;
         $pendingI = pendingI - \{i_{old}\}$ .
    If  $i_{old} \in executingI$ 
        Reduce the priority of  $i_{old}$  to removalPriority.
If  $i_{old} \in intentions$  and ( $urgencyAffected(i_{new}, i_{old})$ )
    Change the priority of  $i_{old}$  or  $i_{new}$  to a higher/lower value or suspensionPriority.
Inform IS the priority of an intention has changed.

```

Fig. 7. Operations of IM.

```

LOOP:
1. Scheduling intentions according to the priorities of intentions.
LOOP1:
     $P_k = \max_{i_j \in pendingI} (P_j)$ ;
    If  $P_k$  equals null or suspensionPriority
        break LOOP1.
     $K = \{intention\ k\ and\ its\ peers\ intentions\}$ ;
    If there is no enough free PE(s) for intentions in K
         $n$  is the number of PEs needed if K can get executed;
         $p_i = \max_{i_j \in I} (P_j)$ 
        where  $I = \{n\ intentions\ in\ executingI\ with\ the\ lowest\ priorities,\ note\ that\ priority\ of\ peer\ intention\ have\ the\ same\ priority\}$ .
        If preempt( $P_k, P_i$ )
             $pendingI = pendingI \cup I$ ;
             $executingI = executingI - I$ ;
            Send interrupts to the PEs to suspend the execution of intentions in I.
        else
            break LOOP1.
    If there is enough free PE(s) for intentions in K
         $pendingI = pendingI - K$ ;
         $executingI = executingI \cup K$ .
        Send interrupts to the PEs to start execution of intentions in K.
2. Handling the completion of the executing of an intention by a PE.
    For each intention  $e \in executingI$ 
        If the executing of  $i_e$  is finished in a PE:
             $executingD = executingD - \{d_e\}$ ;
             $executingI = executingI - \{i_e\}$ ;
            If  $i_{old} \in pendingI$  and  $clash(i_e, i_{old})$ 
                Increase the priority of  $i_{old}$  such that  $i_{old}$  may be considered for execution.
3. Updating the priorities of intentions.
    For each intention  $i \in intentions$ 
        Update the priority of  $i_i$  according to IM instructions or by other factors, like time.
        If priority of  $i_i$  equals removalPriority

```

```

executingD = executingD - { di };
intentions = intentions - { ii }.
Send an interrupt to the PE to remove ii
If priority of ii equals suspensionPriority and ii ∈ executingI
executingI = executingI - { ii and its peers intentions };
inactiveI = inactiveI ∪ { ii and its peers intentions }.
Send an interrupt to the PE to suspend ii and its peers intentions.

```

4. Checking predecessors.

For each intention $i \in intentions$

If $i_i \in pendingI$ and a new predecessor is added to it
 $pendingI = pendingI - \{ i_i \text{ and its peers intentions } \};$
 $inactiveI = inactiveI \cup \{ i_i \text{ and its peers intentions } \}.$

If $i_i \in executingI$ and a new predecessor is added to it
 $executingI = executingI - \{ i_i \text{ and its peers intentions } \};$
 $inactiveI = inactiveI \cup \{ i_i \text{ and its peers intentions } \}.$

If $i_i \in inactiveI$ and all predecessors of it and its peers intentions are completed or removed
 $inactiveI = inactiveI - \{ i_i \text{ and its peers intentions } \};$
 $pendingI = pendingI \cup \{ i_i \text{ and its peers intentions } \}.$

Fig. 8. Operations of IS.

With this framework, an agent developer just needs to concentrate on specifying domain specific definitions of obsolete desires, urgent desires and clashing desires. The desires and intentions will be activated, deactivated, suspended or removed automatically.

2.4 Synchronization among peer intention plans

With multiple PEs, the agent is able to carry out multiple actions simultaneously. Some actions need to be synchronized and some are completely asynchronous. Synchronization is needed among peer intention plans and we classify the synchronization into the following three forms.

- 1) Time stepped synchronization. The execution of intention plan P in one PE has to be at the same 'speed' as those of several other intention plans in several other PEs. IS can play the centralized coordinator in controlling the PEs.
- 2) Barrier style synchronization. The execution of several intention plans in several PEs may progress asynchronously until a certain 'barrier' then they synchronize with each other and continue. This can be managed in the same way as the barrier synchronization in parallel discrete event simulation. Again, IS can coordinate.
- 3) Execute intention plan P1 in PE₁ while intention plan P2 is being executed in PE₂. IS start P1 and P2 in the 2 PEs at the same time and it will terminate P1 in PE₁ as soon as P2 in PE₂ has completed execution.

2.5 What a Parallel BDI Agent Means

A parallel agent architecture has changed the nature of the BDI agent and what agents may expect from other agents in a multi-agent system. In a parallel agent, the belief manager, the intention generator and the intention executor run continuously and all the components in them work in parallel. The interrupt mechanism ensures that an emergency can be dealt with first. Thus, the agent obtains the ability of quick reaction to emergencies and the capacity for careful deliberation when required. With the parallel components, the agent can handle several matters at once. The agent is also able to 'change his mind' towards his desires/intentions. The main differences between a parallel agent and other agents are

- 1) The agent has very high caution level because it is able to monitor the environment continuously. This is not possible with a sequential agent as it needs to switch its processing power among perception, deliberation and execution of intention plans.
- 2) The agent is now more alert to changes in the environment and be able to react faster than sequential agents because the occurrence of an urgent event will lead to interruption to the current operations of the parallel running components. In a sequential agent, it has to wait till the end of the time slice of the currently running component.
- 3) The agent's capability is extended into areas which a sequential agent will find hard if not impossible to manage. A parallel agent is able to get information continuously from different types of sensors through multiple Environment Monitors. It is able to control different actions through multiple Plan Executors. The possibility of cooperation

among several PEs will further enhance the agent's capability and efficiency.

- 4) There will be more uncertainties among the agents in terms of when an agent can expect another agent to get a specific task finished or in what order the tasks given to an agent will be finished. This is the result from multiple tasks being executed in the agent at the same time. Uncertainties will increase if the PEs have different capabilities.

In the next section, we give an example of a software agent built under this framework.

3 A Vessel Captain Agent

We present a software agent of vessel captain navigating in sea using the general framework of parallel BDI model. When a vessel captain is navigating in sea, he has to watch the sea, make decisions and issue commands to the sailors in real time. He has to prioritize his deliberations and commands in a suitable sequence to ensure safety of the vessel. We will show that the parallel agent architecture satisfies the functional requirements of such an agent. We will first show the detailed agent architecture. Then the experimental results are presented and explained.

3.1 Architecture

When an agent is simulating a certain physical system like a human being, the parallel agent under the general framework should be configured such that it has

the same number of Environment Monitors (EMs), Plan Generators (PGs) and Plan Executors (PEs) as the number of parallel physical devices that exist in the physical system to perform the corresponding functions. Fig. 9 shows the detailed software agent architecture for a vessel captain navigating a vessel in the sea. In this architecture, there is just one PG in the agent. This is to simulate the human behaviour that at any time, we deliberate on or think about one matter. To keep the experiment simple, the agent's actions only include vessel maneuvering so only one action can be taken at one time. This results in only one PE in the agent. The number of EMs is also one for the collection of information about its surroundings by the vessel. The link from the Intention Scheduler to EM is to inform it that a certain intention plan has been completed.

Threads are used to simulate the parallel processing elements in the agent. In Fig. 9, these threads are shown in oval and identified with numbers. For example, the belief manager consists of two threads, numbered by 1 and 2. Similarly, the intention generator and the intention executor are made of several threads of their own. Two more threads, the action effector (7) and the interface thread, do not belong to any of the three main components of the BDI agent. The Interface thread will receive commands through user input and send the commands to the belief manager. The action effector thread will execute output physical actions from the intention executor. In our simulation, this action effector thread is responsible for updating the world map with

Fig. 9. Example architecture for a vessel captain agent.

the new position of the vessel which is calculated using physical motion laws. All the threads run in parallel. Message buffers in the three components are used to save the incoming messages from one processing element to another. The messages in these buffers are sequenced by their priorities so that high priority messages are handled first.

The PG and PE are not explicitly shown in Fig. 9 because the plan generation of each desire and the plan execution of each intention are implemented as separate threads. This makes it very easy to manage the suspension and the subsequent resuming of the plan generation of a desire in the PG and that of the plan execution of an intention in the PE. If a thread is suspended, the current working status of the thread is saved in computer memory automatically for resuming the thread later on. The Desire Scheduler and the Intention Scheduler will schedule the running of the threads according to the priorities of the desires and intentions respectively. At any time, just one plan generation thread and one intention execution thread can be activated.

3.2 Experiment

Table 1. Levels of priorities for new beliefs

Level	Explanation	Example
1. low priority	Something the agent needs to deliberate on when it has time	Plan path for future targets
2. medium priority	Something the agent needs to deliberate on not immediately but some fixed time in future	Navigate to the current target
3. high priority	Something the agent needs to deliberate on immediately but still can take time to think carefully	New target/map changed: the agent needs to calculate path
4. very high priority	Something the agent needs to deliberate on immediately and try to make decisions as soon as possible and act	Nearby obstacle: the agent needs to avoid collision immediately

In our simulation experiment, a vessel agent can detect the world information through sensors and the output actions consist of commands of changing translational and rotational speeds. Vessel agents are created with their specific physical parameters. The vessel agent's targets are given when the vessel is created. User can also give commands to the vessel at any time through the command input window. These commands are read by the Interface thread and include a change of the current target or the addition of a new target. The objective of the agent is to navigate to the targets safely and quickly. An example is shown in Fig. 10. In order to achieve its objective, the vessel agent

Fig. 10. Vessel navigation.

should have the abilities to plan its path, to output actions which make it travel to the target, and to avoid obstacles. These behaviours are triggered by new beliefs of the agent. The corresponding plans are worked out in response to the new beliefs. To illustrate the agent's ability to handle matters of different urgencies, four levels of priorities for the vessel agent's beliefs are assumed in the experiment and shown in Table 1.

A navigation example is shown in Fig. 10. The area of interests is shown in a two-dimensional rectangular sea area. There are four islands shown by the shaded areas in the map. As shown in Fig. 10a, the vessel agent at starting point (0,0) is given four targets 1, 2, 3, and 4 to move to. Then a change of plan, that is, new targets 1 and 2, as shown in Fig. 10b are given when the vessel is moving to the old target 1. Then the vessel is able to change its plan and navigate to the new targets. The behaviours of the corresponding threads are recorded. Only the most relevant activities are shown in Table 2. These activities are carried out by the belief manager, intention generator and the intention executor of the agent as stated before.

The sequence of the activities as shown in Table 2 illustrates the behaviour of the parallel agent.

1. The new beliefs of the agent are processed in the order of priority. When the agent is started, new beliefs of the need to go to target 1 followed by 2, 3 and 4 are received. After the beliefs are sent

by the belief generator, the desire generator creates the plan generation threads 0 and 1 (events 493, 494) and the desire scheduler starts the plan generation thread 0 (event 495) while the plan generation thread 1 stays in the desire queue (pending desire). The plan generation thread 1 is something the agent needs to deliberate on when it has time as shown by event 498.

2. The agent is able to carry out several activities at once. As shown by event 498 and 499, the agent is executing plan generation thread 1 and intention thread 0 simultaneously, that is, the agent is processing the belief of the need to go to target 2, 3, 4 and the intention of planning the path to target 1 at once.
3. When there is a more urgent intention, the intention with low priority can be suspended and resumed later. As shown by events 509, 510 and 515, intention 2 is one of the steps in following the path plan to go to target 1 while intention 1 is

to compute the path to target 2, 3, 4. Intention 2 preempts intention 1.

4. When an intention becomes obsolete because of some new beliefs, the agent is able to stop and remove it. As shown by event 534, intention 1 which is being executed by the intention executor is stopped and removed. This is caused by an interrupt from the intention generator when it is processing the belief that the vessel should change course and go to the new targets 1 and 2.
5. The agent can respond to circumstance changes rapidly. Event 523 shows the agent is moving to the old target 1. Event 543 shows the agent is processing the new belief that the target is changed. Event 569 shows the processing of the belief that vessel path is updated in response to the new targets. Event 572 shows the vessel is changing course when it is moving to the old target 1.

Table 2. Processing records of the vessel agent

ID	Insert_Time	Type	Behaviour
492	7:32:20 PM	Agent1	Start!
493	7:32:20 PM	plan generation thread0	Created. Status: New Target
494	7:32:20 PM	plan generation thread1	Created. Status: Idle
495	7:32:21 PM	plan generation thread0	Start. Status: New Target
496	7:32:21 PM	plan generation thread0	Finished. Status: New Target
497	7:32:21 PM	plan execution thread0	Created. Plan execution: 0/1.
498	7:32:21 PM	plan generation thread1	Start. Status: Idle
499	7:32:21 PM	plan execution thread0	Start. Plan execution: 0/1.
...			
504	7:32:23 PM	plan execution thread1	Start. Plan execution: 10.
...			
507	7:33:06 PM	plan generation thread2	Start. Status: The global path updated
508	7:34:38 PM	plan generation thread2	Finished. Status: The global path updated
509	7:34:55 PM	plan execution thread2	Created. Plan execution: 2/3.
510	7:34:56 PM	suspend plan execution1.0	2.2 Start.
511	7:34:56 PM	suspend plan execution1.0	2.2 Suspended.
512	7:34:57 PM	plan execution thread2	Start. Plan execution: 2/3.
513	7:34:57 PM	Subgoal reached	(0,0)
514	7:34:57 PM	plan execution thread2	Finished. Plan execution: 2/3.
515	7:35:05 PM	resume plan execution	plan execution1
...			
523	7:35:57 PM	Action	Created. action: Accelerate to max speed.
...			
534	7:37:40 PM	Remove obsolete plan execution	1
535	7:37:40 PM	plan execution thread5	Created. Plan execution: 2/3.

536	7:37:40 PM	plan generation thread6	Created. Status: Targets changed
537	7:37:40 PM	plan generation thread7	Created. Status: New Target
538	7:37:40 PM	plan generation thread8	Created. Status: Idle
539	7:37:39 PM	plan execution thread4	Start. Plan execution: 4/5.
540	7:37:40 PM	Action	Created. action: Decelerate to min speed.
541	7:37:40 PM	plan execution thread4	Finished. Plan execution: 4/5.
542	7:37:39 PM	plan generation thread5	Finished. Status: Subgoal reached
543	7:37:41 PM	plan generation thread6	Start. Status: Targets changed
...			
568	7:40:16 PM	plan generation thread11	Start. Status: The global path updated
569	7:40:17 PM	plan generation thread11	Finished. Status: The global path updated
570	7:40:13 PM	plan execution thread9	Start. Plan execution: 4/5.
571	7:40:17 PM	plan execution thread11	Created. Plan execution: 2/3.
572	7:40:17 PM	Action	Created. action: Decelerate to min speed.

The experimental results show that the agent built with this architecture has the ability of deliberating on new beliefs and executing current intentions at once. The agent also has the ability to prioritize the deliberations and intention executions so it is able to respond quickly to circumstance changes and all the thinking and acting are done in the appropriate order. The next section will be evaluating the performance of a parallel BDI agent by comparing it with sequential BDI agents.

4 Comparison between the parallel BDI model and the sequential ones

In this section, we evaluate the performance of the parallel agent by comparing it with the sequential agents. There are 3 main or coarse computational components in a BDI agent, the belief manager, the intention generator and the intention executor. In a sequential agent, only one computational component is running at any time. However it is possible to control and manage the 3 components in several different ways in an attempt to get better performance from a sequential agent. On the parallel BDI agent side, under the general framework the maximum parallelism can be realized by having all the processing elements like EMs, BG, DG and so on running in parallel. To demonstrate that parallel BDI agents constructed according to the general framework is able to offer the benefit of parallel actions, we use a small parallel agent where only the 3 main components, the belief manager, the intention generator and the intention executor operate in parallel. This means the processing elements in the same component will run sequentially. We first describe the five sequential BDI agents each with their own way of controlling and managing their computational

components. Then the simulation experiments are presented and the results are analyzed

4.1 Sequential BDI agents

- (1) As shown in Fig. 11a, the 3 components run in a cyclic way and each uses up the pre-allocated and fixed time resource. The deliberation/intention cannot be suspended and resumed. If the remaining time of a component (only the *deliberate* and the *execute* components) is not sufficient for a deliberation/intention to be finished, the remaining time will be wasted.
- (2) This is a variant of agent 1. It suspends a task when the time allocated to the current component is used up and resumes it when the component's turn comes in the next cycle. For example, the *execute* action can start an *intention* which costs 5 time units when there is only 1 time unit remaining.
- (3) A more flexible way is to allocate time resources to the *deliberate* and *execute* components only when needed. If a component has nothing to do, it terminates and the next component starts. In order to keep the agent vigilant, the *detect* component always uses up all its allocated time. The actual time used for *deliberate/execute* should not exceed the maximum pre-allocated time to such a component. This agent has a cycle time ranging from a minimum that is the fixed time for the *detect* component to a maximum that is the sum of the allowable time window lengths of the 3 components. The tasks cannot be suspended.
- (4) Different from agent 3, the tasks can be suspended.

- (5) This agent has a cycle as shown in Fig. 11b, in each cycle, after the *detect* component, the agent will choose to *deliberate* or *execute* based on the maximum priority of *deliberations* and *intentions*. After *deliberate/execute* is finished, another cycle begins. This makes the agent more watchful for emergencies.

Fig. 11. Sequential BDI agents.

The characteristics of the five sequential BDI agents are summarized in Table 3. In all the sequential agents, when there are more than one deliberations/intentions to handle in the *deliberate/execute* component, the one with highest priority will be processed first. The performance of these agents will be compared with the parallel BDI agent.

Table 3. Sequential agents

Agent no	Flexible time allocation?			suspend - resume?	Illustration
	Detect	Deliberate	Execute		
1	N	N	N	N	Fig. 11(a)
2	N	N	N	Y	Fig. 11(a)
3	N	Y	Y	N	Fig. 11(a)
4	N	Y	Y	Y	Fig. 11(a)
5	N	Y	Y	Y	Fig. 11(b)

Table 4. Allocation schemes

Configuration	detect	deliberate	execute
C1	1	4	10
C2	3	4	8
C3	5	3	7

With different time allocation schemes for the three components, the sequential agent will show different characteristics and performances. In the experiments, we used three time allocation schemes for sequential

agents reflecting different emphasis on each of the three components. For a maximum cycle time of 15 time units, three schemes showing the fixed or maximum allowable time quota for each component of the BDI agent are given in Table 4.

The sequential agent with configuration C1 puts more emphasis on executing intentions with the risk of overlooking emergencies. C3 gives more time to the *detect* component to be more vigilant. But the time for the *execute* component is cut. C2 is a compromise between C1 and C3.

Each sequential agent described in Table 3 will be configured according to C1, C2 and C3 respectively in the experiments to compare them with the parallel agent.

4.2 The Input Data

The evaluation of the sequential and parallel agents is done by simulation of the processing of events by agents. All the sequential agents and the parallel agent will process the same sequences of events. Each event will be processed by the 3 computational components of the BDI agent, namely, the belief manager (*detect*), the intention generator (*deliberate*) and the intention executor (*execute*). To evaluate the agent ability to handle events of different importance or urgencies, events will have one of the four different priority levels 1 to 4, with 4 being the highest. We assume that an event can be detected and a belief generated in 1 time unit and each deliberation to generate an intention plan takes 1 to 3 time units. The execution times of the intention plans of events at all priority levels is uniformly distributed in the range from 1 to 7 time units. So the average deliberation execution time is 2 time units and the average intention execution time is 4 time units. This also means the average time required to handle an event is 7 (that is, 1+2+4) time units.

We use the exponential density function to represent the inter-arrival time between any two events. As shown in [34], the exponential density function is memoryless and often used to model lifetimes or waiting times. The cumulative distribution function is shown as:

$$\text{cdf}(t) = 1 - e^{-\lambda t} \quad (1)$$

where, $1/\lambda$ is the mean; t is the time units.

Table 5. Events statistics

Set	Expected average interval $1/\lambda$	Events count					Actual average interval	Average deliberation time	Average intention time
		Priority				sum			
		1	2	3	4				
a	4	262	250	235	251	998	4.48	1.99	3.96
b	7	240	260	247	247	994	7.31	1.97	4.08
c	15	268	244	239	234	985	14.58	1.98	3.89

$\text{cdf}(t)$ shows the probability that the inter-arrival time between 2 events is less than t . So given a total number of events, sum , to be used in our experiments, we can decide the number of the intervals with length of n time units by:

$$G(n) = (\text{cdf}(n) - \text{cdf}(n-1)) * \text{sum} \quad (2)$$

The intervals are kept in a vector. Then the intervals are selected randomly. The current time plus the interval length is the arrival time of the next event. The event priority is selected randomly from 1 to 4.

We consider three sequences of events with different average inter-arrival times. The average inter-arrival times of the 3 sequences of events are respectively smaller than, equal to and larger than the average processing time required by an event. The events statistics used in the experiments are shown in Table 5.

When an event arrives and the agent is not doing detection, the event will be stored in an event buffer. The agent can retrieve the new events later. After receiving the event, the agent will create one *deliberation* for it. The *deliberation* will be added in a deliberation queue. Each *deliberation* and the *intention* plan selected by the deliberation will have the same priority as the event.

4.3 Comparison Results and Analysis

We will use the *response time* to evaluate how well the agent processes the event. The *response time* is

defined as the time between the arrival of the event and end of the execution of the intention plan chosen for the event. The response time is calculated as the sum of the time for detecting the event, deliberation and execution. The overhead in transmitting messages between the three parallel components in the parallel agent is ignored as it is felt that the delay in passing the interrupts is very small. This is justifiable if we restrict the scale of the parallel BDI agent such that the parallel components can be run in multiple processor cores on a single chip. For example, the Sun Niagara processor has 8 complete processor cores that share a bus to memory. Each core has 4 hardware threads. The communication delay between the parallel components of the agent running on a single chip of such kind of platforms is negligible if compared with communication through a local network.

The results of the experiments are presented in Table 6. The ART is the Average Response Time of all events. ART_p stands for the ART of the events with priority p . ART_w is the weighted ART by the priorities of the events.

$$\text{ART} = \frac{\sum_{i=1}^{\text{Count}} T_i}{\text{Count}}, \quad \text{ART}_w = \sum_{p=1}^4 \frac{\text{ART}_p * p}{1 + 2 + 3 + 4} \quad (3)$$

where, T_i is the response time for event i ; Count is the number of events.

Table 6. ART of the events by the agents

Set	Config	Agent no	ART_p				ART	ART_w
			1	2	3	4		
a	C1	1	3731.59	1577.26	90.22	22.41	1401.62	724.64
		2	4443.1	1621.04	64.09	23.66	1593.54	797.21
		3	2706.14	927.34	58.8	19.54	961.49	481.54
		4	3300.26	635.3	42.51	20.86	1040.8	478.18
		5	3318.06	651.88	38.66	20.87	1048.72	482.13
	C2	1	4416.69	2114.99	144.25	23.12	1729.08	917.19
		2	4581.51	1818.47	73.33	26.18	1682.14	854.32
		3	4046.55	1735.6	115.54	22.51	1529.96	795.44
		4	4496.36	1599.33	71.64	25.2	1604.25	801.07
		5	4531.08	1651.86	63.06	23.13	1623.98	811.65
	C3	1	6523.12	3832.43	656.97	27.9	2834.22	1627.05
		2	6621.4	3990.95	573.05	32.56	2881.14	1645.27
		3	5959.58	3193.37	371.31	23.75	2457.89	1355.53
		4	6017.61	3415.7	252.62	28.57	2502.08	1372.12
		5	6112.63	3523.51	312.36	26.88	2567.67	1420.42
	Parallel		48.81	17.59	11.18	7.75	21.8	14.85
b	C1	1	961.05	78.29	28.69	18.4	264.22	127.73
		2	456.77	50.65	26.58	20.26	135.17	71.88
		3	68.91	26.33	18.15	13.45	31.38	22.98
		4	65.3	27.65	19.14	15.74	31.67	24.1
		5	73.74	28.82	20.72	15.81	34.42	25.68

	C2	1	2177.75	132.21	35.53	19.48	574.07	262.67
		2	1075.42	73.53	33.39	23.32	292.98	141.59
		3	1051.26	86.14	29.45	17.5	288.02	138.19
		4	903.78	78.73	34.39	23.95	253.31	126.02
		5	1015.03	62.61	31.53	21.31	274.58	132.01
	C3	1	4520.94	728.04	57.62	19.52	1301.18	622.8
		2	4814.97	408.03	44.98	24.77	1286.63	586.5
		3	3688.76	459.78	45.62	18.09	1026.74	481.75
		4	4109.05	208.17	39.25	23.95	1062.28	473.89
		5	4590.59	267.35	37.85	22.12	1193.23	532.73
	Parallel		13.12	10.0	8.58	7.35	9.74	8.83
c	C1	1	34.46	25.52	20.15	16.4	24.48	21.16
		2	27.16	22.91	19.26	16.65	21.69	19.73
		3	11.19	9.73	8.79	9.03	9.73	9.32
		4	11.11	9.61	8.96	9.18	9.76	9.39
		5	14.5	12.14	10.41	9.59	11.76	10.84
	C2	1	44.23	28.39	21.44	16.93	28.29	23.3
		2	31.07	23.59	20.99	17.61	23.57	21.16
		3	17.63	13.29	11.49	10.69	13.41	12.14
		4	16.19	13.64	11.82	11.09	13.28	12.32
		5	27.78	20.31	16.24	13.88	19.83	17.27
	C3	1	76.7	34.8	22.0	15.69	38.55	27.5
		2	51.63	31.1	22.37	17.95	31.44	25.27
		3	35.13	20.25	14.54	12.06	20.97	16.75
		4	29.42	19.42	15.31	13.85	19.82	16.96
		5	41.46	26.29	18.28	15.89	26.0	21.24
Parallel		8.92	7.75	7.52	6.87	7.8	7.45	

Parallel agent vs sequential agents

We should note that the parallel agent uses 3 times CPU time as the sequential agents do because the belief manager, the intention generator and the intention executor are running in parallel. However, the response times of the parallel agent and the various sequential agents in Table 6 (rows with agent nos. 1 to 5), show that the parallel agent responds more than 3 times faster than the sequential agents in many cases. This shows that in these cases, by providing the parallel agent with CPU time which is 3 times that of a sequential agent, we are able to cut the average response time by more than two third of the sequential agent's. Especially for the events sets *a* and *b*, where the events arrive faster than the processing ability of the sequential agents, the sequential agent is not able to cope. This clearly shows that in such environments, the sequential agents are not suitable.

One thing to note is that in a few cases the parallel agent responds less than 3 times faster than the sequential ones. This happens with sequential agents 3, 4, 5 when the event sequence is set *c*. The ARTs and the weighted ARTs of agents 3, 4 with configuration C1 and C2 are less than 2 times those of the parallel agent. This means when the inter-arrival time is long enough, the parallel agent will not show its advantage. Because the average inter-arrival time is 14.58, the sequential

agents have sufficient time to process the incoming events. Though configuration C3 is allocating too much time to the *detect* component of the agent, the performance of the sequential agents with C3 is not too bad. However, the parallel agent is still valuable to process the real emergencies with the highest priority. As seen in Table 6, ART₄ to process the events set *c* is 6.87 for parallel agent and 9.03 for the sequential one with best performance.

The priority 4 events are the highest priority events in the experiments. The average time needed to process one such event in the ideal case (ATN₄) is calculated by:

$$ATN_4 = \frac{\sum_{i=1}^{Count_4} (TDetection_i + TDeliberation_i + TExecution_i)}{Count_4} \quad (4)$$

where, $TDetection_i$ is the time used to detect the event *i*; $TDeliberation_i$ is the time used to deliberate the event *i*; $TExecution_i$ is the time used to execute the plan generated based on the event *i*; $count_4$ is the number of events with priority 4.

In this experiment, $TDetection$ equals to 1. $TDeliberation$ and $TExecution$ are shown in the events list in Table 5. Using the equation, ATN₄ equals to 7 for set *a*, 7.02 for set *b*, 6.79 for set *c*. Compared to the ART₄ (column 7 in Table 6) we can see that the parallel

agent spends just 0.75 time units more for set *a*, 0.33 time units more for set *b* and 0.08 time units more for set *c*. Set *a* is a sequence of events with an inter-arrival time smaller than the processing time required for an event. So the intention generator and the intention executor are busy handling other beliefs and intentions when a high priority belief/intention comes to them. Here the interrupt mechanism in the parallel agent is able to guarantee immediate handling of higher priority items while the sequential agents are not able to do this. It is clear that the parallel agent has a big advantage over the others on processing real emergencies.

For events with lower priority, the difference of ART between the sequential agents and the parallel agent is bigger.

Our conclusion in the comparison between the parallel agent and the sequential ones is that the parallel agent is confirmed to perform better when the event inter-arrival time is smaller than the processing time of the events. When the average inter-arrival time of events is larger than the processing time of the events, the sequential agents will on average still be slower than the parallel agent, even though not more than 3 times slower. This is because the sequential agents will show poor performance whenever the inter-arrival time between any 2 events is smaller than the processing time of the first event. This can still happen when the average inter-arrival time is larger than the average event processing time. When it is important to react in time, the parallel agent is the only one able to guarantee performance. The ability to react fast to important and urgent events is also required when the agent is to demonstrate human-like intelligent behaviour.

With different time resources allocation in sequential agents

Looking at the ART and ART_w columns and comparing the corresponding rows for configurations C1, C2 and C3, we conclude that the performances of the sequential agents with configuration C3 are significantly worse than those with configuration C1 and C2. This shows that the sequential agents perform badly if they spend more time on detecting and less time on deliberation and intention execution. The processing of emergencies is often postponed, though the emergencies are detected earlier in configuration C3. This can be seen that in most cases the processing of the highest priority events also have longer response time. This indicates that in real life, the agent is not reacting to high priority events quickly and is taking a longer time to react to other events.

We also observe that the performances of the sequential agents with configuration C1 is significantly better than C2. This shows that the sequential agents

perform much better if they spend short time on detecting and more time on deliberation and intention execution. Because the deliberating and executing components get more time resources, the beliefs and intention plans get cleared faster so the events experience shorter response time.

With different ways of controlling the computational components in sequential agent

Looking at the ART and ART_w columns and comparing the corresponding rows among the sequential agents, we see that agent 1 and 2 are the losers in all cases. This is expected because of their rigid way of controlling the *detect*, *deliberate* and *execute* components. In the best performing configuration C1, agents 3, 4 and 5 have comparable performance in all event sequences a, b and c. So we conclude that if a component has nothing to do, it is better to give way to the next computational component.

For set *b* and *c* in configurations C2 and C3, generally agents 3 and 4 are better with 4 being the best more often but agent 5 is not the loser every time. We may conclude that it is better to allow the computational components to start processing an item if there is one, then suspend it when its time quota is up and resume in the next round. But this policy is not always effective. At first, it is interesting to notice that the ART₄ is normally better for the agent 3 than agent 4 (Just one exception is found: set *a* Configuration C1). This is because the policy in agent 3 makes it keep on processing a lower-priority task when a higher-priority event arrives. The sequential agent has no interruption schema, so the new event will not be handled immediately. In Table 6, we can see that the ART₄ is from 1 to 5 time units bigger for agent 4 than agent 3. This also affects the ART₃ and ART₂ in some cases. Another interesting phenomenon is that for set *a*, the ARTs are mostly worse for the agents with the suspend-resume mechanism (2, 4) than the agents without such mechanism (1, 3). It is because in agents 1 and 3, the execution of the tasks is not strictly scheduled according to the policy that the higher-priority tasks get processed first. If the current remaining time-slot is not sufficient to execute a task with high-priority, the low-priority task with suitable execution time will be processed earlier. Then the tasks with shorter execution time have more chances to be processed first. For a high-density events set, as the set *a*, such fact will decrease the average waiting time of the tasks. So ART is increased.

5 How much parallelism

In Section 3, we propose that when an agent is simulating a certain physical system, the parallel agent should be configured such that it has the same number of Environment Monitors (EMs), Plan Generators (PGs) and Plan Executors (PEs) as the number of parallel physical devices that exist in the physical system to perform the corresponding functions. In this section, we consider agents that are not simulating a physical system. In this case, the constraint is the parallelism that can be supported by the physical computer. For example, if a computer has 2 CPUs, only 2 processing elements can be running in parallel. However, given a fixed number of CPUs, there is still the issue of how to distribute the CPU power to the processing elements. For example, 6 CPUs can be used by the PGs and PEs, how do we decide whether to have 2 PGs and 4 PEs or 3 PGs and 3 PEs? In this section, we present experimental results on agents with different configurations based on the general parallel BDI agent framework. The agents will process the same sequences of events. The event sequences are designed in the same way as in the last section. The statistics about the sequences used are shown in Table 7.

For simplicity, we assume there is only one type of

sensor input so there is just one EM (K_I in Fig. 2 is 1). The following are assumed in the experiment:

1. The incoming event can be detected and beliefs updated in 1 time unit. So processing time in BG is 1 time unit.
2. The DG takes 1 time unit to generate a desire.
3. Each deliberation to generate an intention takes 1 to 3 time units, uniformly distributed. This is the processing time in PG. So the expected average processing time in a PG is 2 time units.
4. The intention execution time of events at all priority levels is uniformly distributed in the range from 1 to 7 time units. This is the processing time in PG. The average is 4 time units.
5. The times used in DS, IM, IS are significantly shorter than the processing times in other processing units like BG, DG, PG etc and can be ignored.
6. The overhead of transmitting messages between the various processing units in the parallel agent is not included as it is assumed that the agent is running on a machine with multiple processors sharing memory. So the delay in passing the interrupts is very small.

Table 7. Events statistics

Set	Expected average interval $1/\lambda$	Events count					Actual average interval	Average PG time	Average PE time
		Priority				sum			
		1	2	3	4				
a	2	244	286	234	235	999	2.54	1.99	4.16
b	4	270	237	255	236	998	4.48	1.99	4.01
c	7	235	244	264	251	994	7.31	2.01	3.98

Table 8. ART of the events by the agents

Set	K_2	K_3	ART _p				ART	ART _w
			1	2	3	4		
a	1	1	2364.78	813.83	18.64	9.26	817.12	408.54
	1	2	22.32	11.8	8.89	8.17	12.84	10.53
	2	2	18.21	10.28	8.4	8.06	11.25	9.62
	2	3	9.12	8.59	8.01	8.01	8.45	8.24
	2	4	8.49	8.35	7.99	8.01	8.22	8.12
	3	3	9.0	8.55	8.0	8.01	8.41	8.21
	3	4	8.4	8.28	7.97	8.01	8.17	8.09
b	1	1	49.33	17.36	11.3	8.84	22.45	15.33
	1	2	9.77	9.07	8.36	8.34	8.91	8.64
	2	2	8.87	8.52	8.13	8.28	8.46	8.34
	2	3	7.99	8.03	7.96	8.28	8.06	8.1
	2	4	7.84	8.0	7.96	8.28	8.01	8.08
	3	3	7.96	8.04	7.96	8.28	8.05	8.1
	3	4	7.82	7.99	7.96	8.28	8.0	8.08
c	1	1	14.35	10.84	9.85	8.31	10.77	9.88
	1	2	9.02	8.2	8.27	8.05	8.37	8.24
	2	2	8.47	7.97	8.09	8.01	8.13	8.07
	2	3	8.12	7.81	8.05	8.0	8.0	7.99
	2	4	8.08	7.81	8.05	8.0	7.99	7.99
	3	3	8.11	7.81	8.05	8.0	7.99	7.99
	3	4	8.07	7.81	8.05	8.0	7.98	7.98

All these assumptions mean the expected average time required to process an event is 8 (1+1+2+4) time units.

The results of the experiments with different combinations of $K2$ and $K3$ devices (refer to Fig. 2) are presented in Table 8.

The priority 4 events are the highest priority events in the experiments. The average time needed to process one such event in the ideal case (ATN_4) is calculated as the sum of the detection time (1), desire generation time (1), average PG time and average PE time for all the events with priority 4. In the events sequence used, ATN_4 equals to 8.01 for set *a*, 8.28 for set *b*, 8.0 for set *c*. Compared to the ART_4 in Table 8 we can see that the parallel agent spends just a little more or the same amount of time for processing the events with priority 4. This confirms that the interrupt mechanism in the parallel agent is able to guarantee immediate handling of higher priority items.

We will examine the waiting times for deliberation and execution in 7 different configurations of the parallel agent. Waiting time for deliberation (AWTD) means the time a desire spent in *pendingD* waiting. This includes the time before the plan generation is started and the time when the plan generation for this desire is suspended. AWTE, average waiting time for intention execution, is defined in a similar way. A shorter waiting time means that the processing is quicker.

The AWTD is only related to $K2$, the number of PGs. Table 9 confirms that with a larger $K2$, the deliberations can be finished quicker. In the three sets of environments, *a*, *b* and *c*, 3 PGs are enough to provide the agent the ability to deliberate any event immediately after it is received.

Table 9. Average waiting time for deliberation

Set	$K2$	AWTD _p				AWTD
		1	2	3	4	
<i>a</i>	1	5.95	2.13	0.62	0.13	2.24
	2	0.09	0.09	0.01	0.0	0.05
	3	0.0	0.0	0.0	0.0	0.0
<i>b</i>	1	1.26	0.74	0.31	0.07	0.61
	2	0.03	0.01	0.0	0.0	0.01
	3	0.0	0.0	0.0	0.0	0.0
<i>c</i>	1	0.63	0.25	0.19	0.04	0.27
	2	0.01	0.0	0.0	0.0	0.0
	3	0.0	0.0	0.0	0.0	0.0

AWTE is affected by both $K2$ and $K3$ (the number of PEs). The statistics is shown in Table 10. With a same $K3$, the agent with a larger $K2$ can produce intentions earlier. So in such case, the AWTE may be increased. But referring to Table 8, we can see that the total ART is decreased. It is easy to see that with $K2=1$ and $K3$ increased from 1 to 2, the AWTE is greatly decreased. Because in an event-congested environment

like set *a* (interval=2.54), the agent with 1 PE cannot process all the intentions in time.

Table 10. Average waiting time for execution

Set	$K2$	$K3$	AWTE _p				AWTE
			1	2	3	4	
<i>a</i>	1	1	2350.5 2	803.4 6	10.04	1.12	806.74
	1	2	8.06	1.43	0.29	0.03	2.45
	2	2	9.81	1.94	0.41	0.05	3.06
	2	3	0.72	0.25	0.03	0.0	0.25
	2	4	0.09	0.01	0.0	0.0	0.02
	3	3	0.69	0.3	0.03	0.0	0.26
	3	4	0.09	0.03	0.0	0.0	0.03
<i>b</i>	1	1	40.27	8.64	3.04	0.5	13.84
	1	2	0.71	0.34	0.1	0.0	0.3
	2	2	1.05	0.53	0.16	0.01	0.45
	2	3	0.16	0.04	0.0	0.0	0.05
	2	4	0.02	0.0	0.0	0.0	0.01
	3	3	0.17	0.05	0.0	0.0	0.06
	3	4	0.02	0.0	0.0	0.0	0.01
<i>c</i>	1	1	5.65	2.78	1.61	0.27	2.51
	1	2	0.32	0.14	0.03	0.01	0.12
	2	2	0.39	0.16	0.04	0.01	0.14
	2	3	0.04	0.0	0.0	0.0	0.01
	2	4	0.0	0.0	0.0	0.0	0.0
	3	3	0.04	0.0	0.0	0.0	0.01
	3	4	0.0	0.0	0.0	0.0	0.0

The overall performance of the agent is decided by the slowest processing unit of the system. In the parallel agent, plan generation and intention execution are the most time consuming. The balance between $K2$ and $K3$ should be:

$$\frac{K2}{K3} = \frac{AVG_d}{AVG_e} \quad (5)$$

where, AVG_d is the average PG time,

AVG_e is the average PE time.

Tables 8 and 10 confirm that having $K2 = 2$ and $K3 = 4$ is better than having $K2 = 3$ and $K3 = 3$, more so when the arrival rate of events is high. If the average inter-arrival time between events equal to AVG_i , the agent needs $K2 = \frac{AVG_d}{AVG_i}$ PGs to generate plans to

prevent build-up of desires in *pendingD*. In the situation where the arrival of events are very dynamic, that is, the inter-arrival time changes drastically, the best solution will be for the Desire Scheduler (DS) and the Intention Scheduler (IS) to dynamically adjust the number of PGs and PEs in response to the changes in the arrivals of events. In other words, the DS and IS should self organize what is the best ratio of PGs to PEs.

If there is at most 1 event coming in 1 time unit, the maximum values for $K2$ and $K3$ should be smaller than

the maximum PG and PE time respectively. For example, in the experiment, maximum PG time is 3, so 3 PGs are enough to ensure that waiting time for a PG is 0 and AWTD will as a result be 0.

In general, if it is critical to keep the waiting time for PG to zero, the number of PGs required can be calculated by

$$K2 = \lceil b/a \rceil \quad (6)$$

where, a is the minimum time interval between two events and b is the maximum processing time for PG to generate an intention plan for a desire.

This also applies to the value of $K3$ where a is the minimum time interval between the arrival of two intention plans produced by PGs and b is the maximum processing time for PE to execute an intention plan for a desire. With these values for $K2$ and $K3$, whenever a new desire is generated or a new intention plan is generated, there is always a PG or a PE available to process them. It can be stated that the minimum time interval between the arrivals of two intention plans produced by PGs is the smaller value between the minimum inter-arrival time of events and the minimum processing time for PG to generate an intention plan for a desire.

It should be noted that assumptions are made in our study that the times used in DS, IM, IS are significantly shorter than the processing times in other processing units like BG, DG, PG etc and can be ignored. This will be true when the parallel agent is not made of massively parallel processing units. For example, if a robotic agent is equipped with a processor that has 8 complete processor cores that share a bus to memory, the assumption about the DS, IM and IS will be valid. It will not be valid if it is a software agent with a large number of BGs, DGs and PGs. In such cases, it will be necessary to compare whether it is better to have a few agents with massive parallel operations in each, or a higher number of cooperative agents, each with less parallel operations in them.

6 Conclusions

In this paper, a general framework for the BDI model in a dynamic environment is proposed. It is a parallel agent architecture that supports the following agent abilities at architecture level: (1) the ability to monitor the environment at all times and respond to emergencies timely; (2) the ability to reconsider and re-schedule goals, intentions and actions in reaction to unexpected or new information; (3) the ability to perform multiple actions at once; (4) the ability to perceive, deliberate and act simultaneously; (5) the ability to prioritize the deliberations and intention executions. We defined the functions and the operations

of the processing units in the agent and how these units interact, cooperate and synchronize with each other. We discussed what a parallel BDI agent means to a multi-agent system. With the advances in semiconductor technology which allow multiple processing units to be implemented on the same silicon chip, a parallel BDI agent will be an effective way to enable it to perform in constantly changing environments when the arrival rate of events is high.

We illustrate the working of a parallel agent under the general framework with an agent simulating the behaviour of a vessel captain navigating in sea. Then the performance of a parallel agent is evaluated against several versions of sequential agents. The results show that the parallel BDI agent outperforms the sequential ones in offering significantly shorter response time to events with various inter-arrival times. The parallel BDI agent with its interrupt mechanism is able to guarantee fast reaction to high priority events where none of the sequential ones are capable of. The issue of how much parallelism and how to configure a parallel agent based on the general framework are studied by experiments with different configurations of the parallel agent. It is found that under a relative static situation where the demand for deliberation power and that for plan execution power are not changing drastically, the computing power of the agent can be allocated to reflect the demands. But in a more dynamic situation, a dynamic approach will be needed.

This parallel model should be helpful in the research of continual planning. Continual planning means that the agent will be continuously planning, interleaving planning with execution, because its plans can undergo continual evaluation and revision [11]. This is also known as planning and replanning [23]. For a parallel agent, it will be able to continue with planning while executing an intention plan. Several techniques have been proposed for continual planning. For example, the CPEF (continuous planning and execution framework) [27] integrates HTN planning technique [12, 13] to implement open-ended reasoning. Open-ended planning process allows the agent not to generate full level plans before execution so planning can be done in a Just-in-time manner. In UM-PRS [24], the hierarchy of the plans is kept for monitor plan execution and replanning. Obviously, the techniques for continual planning increase the needs for time resource. In order to replan, it is better for the agent to detect new situations frequently. So the parallel architecture is suitable to support continual planning. Furthermore, a high level, abstract intention can be realized by satisfying several sub-intentions and the execution and coordination of the sub-intentions are done by the parallel algorithm. Thus, the intention can be finished quickly and computation resource is utilized efficiently.

Another possible utility of the architecture is that it can provide the agent some adaptive behaviour by combining automatic learning algorithms for some special problems. Being able to adapt to changes is an important attribute for agents to show the autonomy and proactiveness properties [16]. With this ability, the agents can respond to changes in dynamic environments more intelligently. The agents can improve performance continually without human interfere. Many mature learning algorithms have been designed and applied in the machine learning areas [26]. But in the plan-based architectures, such as PRS, it is hard to combine the learning algorithms within the reasoning process. An experimental step was taken in [17]. The learning is implemented by applying the Top-down induction of decision trees on the agent's action models. The models are labeled with success or fail tag. And the models are organized as the decision trees. In the situations with fixed action steps, the agent can interact with environment with past experiences. But for agents working in continuous environment, the limitation is obvious: the models may be too voluminous to save. We have proposed to extend the original BDI model by adding an experience function library [41]. Some complicated algorithms can be coded in this library and called. Combining that extension and the parallel architecture, it is possible to incorporate learning algorithms as experience functions. For example, in a vessel agent, it is possible to implement the obstacle avoidance function with the Q-learning algorithm. The agent can rely on past experience in controlling a vessel and accumulate knowledge and experience as time passes.

References

- [1] S. Ambroszkiewicz and J. Komar, A Model of BDI-Agent in Game-Theoretic Framework, in Proceedings of *Formal Models of Agents, ESPRIT Project ModelAge Final Workshop*, 1997, pp. 8-19.
- [2] E. Amir and P. Maynard-Reid II, Logic-Based Subsumption Architecture, in Proceedings of the *Sixteenth International Joint Conference on Artificial Intelligence*, July 31-August 6, 1999, pp. 147-152.
- [3] E. Amir and P. Maynard-Reid II, LiSA: A robot driven by logical subsumption, in Proceedings of *Working Notes of the CommonSense01 Symposium*, New York, 2001.
- [4] R. H. Bordini and J. F. Hübner, BDI agent programming in AgentSpeak using Jason, in Proceedings of the *Sixth International Workshop on Computational Logic in Multi-Agent Systems (CLIMA VI)*, London, UK, June 27-29, 2006.
- [5] M. E. Bratman, *Intentions, Plans, and Practical Reason*, Harvard University Press, Cambridge, MA, 1987.
- [6] R. A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, **2** (1) (1986). 14-23.
- [7] R. A. Brooks, Elephants Don't Play Chess, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, The MIT press, London, 1991, pp. 3-15.
- [8] R. A. Brooks, Intelligence without Representation, *Artificial Intelligence*, **47** (1991). 139-159.
- [9] M. Dastani, F. d. Boer, F. Dignum, W. V. d. Hoek, M. Kroese and J.-J. Meyer, Programming the Deliberation Cycle of Cognitive Robots, in Proceedings of the *Third International Cognitive Robotics Workshop, held in conjunction with AAAI-2002*, Edmonton, Canada, 2002.
- [10] M. Dastani, F. d. Boer, F. Dignum and J.-J. Meyer, Programming Agent Deliberation: An Approach Illustrated Using the 3APL Language, in Proceedings of the *Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, Melbourne, 2003.
- [11] M. E. DesJardins, E. H. Durfee, J. Charles, L. Ortiz and M. J. Wolverton, A survey of research in distributed, continual planning, *AI Magazine*, **21** (4) (2000).
- [12] K. Erol, J. Hendler and D. S. Nau, HTN planning: Complexity and expressivity, in Proceedings of the *Twelfth National Conf. on Artificial Intelligence*, 1994, pp. 1123-1128.
- [13] K. Erol, J. Hendler and D. S. Nau, *Semantics for Hierarchical Task-Network Planning*, Technical report CS-TR-3239, UMIACS-TR-94-31, ISR-TR-95-9 at the University of Maryland, 1994.
- [14] I. A. Ferguson, *TouringMachines: An architecture for dynamic, rational, mobile agents*, PhD Thesis, Comput. Sci. Lab, University of Cambridge, Cambridge, U.K, 1992.
- [15] I. A. Ferguson, Autonomous agent control: a case for integrating models and behaviors, *Working Notes AAAI Fall Symposium on Control of the Physical World by Intelligent Agents*, New Orleans, LA, 1994, pp. 46---54.
- [16] M. Georgeff, B. Pell, M. Pollack, M. Tambe and M. Wooldridge, The Belief-Desire-Intention Model of Agency, in M. J.P., S. M. and R. A., eds., *Intelligent Agents V: Theories, Architectures, and Languages*, Springer-Verlag, Berlin, 1999.
- [17] A. Guerra-Hernandez, A. El Falla-Seghrouchni and H. Soldano, Learning in BDI Multi-agent Systems, in Proceedings of *CLIMA IV- Computational Logic in Multi-Agent Systems*, Fort Lauderdale, FL, USA, January 6-7, 2004, pp. 218-233.
- [18] B. Hayes-Roth, Architectural Foundations for Real-Time Performance in Intelligent Agents, *Real-Time Systems*, **2** (1990). 99-125.
- [19] N. Howden, R. Ronnquist, A. Hodgson and A. Lucas, JACK Intelligent Agents™ - Summary of an Agent Infrastructure, in Proceedings of the *5th International Conference on Autonomous Agents*, Montreal, Canada, May 28-June 1, 2001.

- [20] F. F. Ingrand, M. P. Georgeff and A. S. Rao, An architecture for real-time reasoning and system control, *IEEE Expert*, **7** (6) (1992). 34--44.
- [21] L. P. Kaelbling, Rex: A symbolic language for the design and parallel implementation of embedded systems, in Proceedings of the *AIAA Conference on Computers in Aerospace VI*, Wakefield, MA, 1987, pp. 255--260.
- [22] K. Kostiadis and H. Hu, A Multi-threaded Approach to Simulated Soccer Agents for the RoboCup Competition, *RoboCup-99: Robot Soccer World Cup III*, Springer-Verlag, London, UK, 2000, pp. 366-377.
- [23] R. v. d. Krogt, M. d. Weerdt and C. Witteveen, A Resource-based Framenwork for Planning and Replanning, *Web Intelligence and Agent Systems*, IOS Press, **1** (3) (2003). 173-186.
- [24] J. Lee, M. J. Huber, E. H. Durfee and P. G. Kenny, UM-PRS: an implementation of the procedural reasoning system for multirobot applications, in Proceedings of *AIAA/NASA Conference on Int. Robots in Field, Factory, Service and Space*, American Institute of Aeronautics and Astronautics, 1994.
- [25] P. Lokuge and D. Alahakoon, A Motivation Based Behavior in Hybrid Intelligent Agents for Intention Reconsideration Process in Vessel Berthing Applications, in Proceedings of the *4th International Conference on Hybrid Intelligent Systems*, Kitakyushu, Japan, December 5-8, 2004, pp. 124-129.
- [26] T. M. Mitchell, *Machine Learning*, McGraw Hill, 1997.
- [27] K. L. Myers, CPEF: A continuous planning and execution framework, *AI Magazine*, **20** (1999). 63-70.
- [28] A. Pokahr, L. Braubach and W. Lamersdorf, A Flexible BDI Architecture Supporting Extensibility, in Proceedings of the *2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2005)*, Compiègne University of Technology, France, September 19-22, 2005, pp. 379-385.
- [29] A. Pokahr, L. Braubach and W. Lamersdorf, A Goal Deliberation Strategy for BDI Agent Systems, in Proceedings of *Third German conference on Multi-Agent System TEchnologieS (MATES-2005)*, Koblenz, Germany, September 11-13, 2005, pp. 82-94.
- [30] O. Rana, M. Winikoff, L. Padgham and J. Harland, Applying Conflict Management Strategies in BDI Agents for Resource Management in Computational Grids, in Proceedings of *Computer Science 2002, Twenty-Fifth Australasian Computer Science Conference (ACSC2002)*, Monash University, Melbourne, Victoria, Australia, January/February, 2002.
- [31] A. S. Rao and M. Georgeff, Modeling rational agents within a BDI architecture, in Proceedings of the *Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, Cambridge, MA, USA, April 22-25, 1991, pp. 473-484.
- [32] A. S. Rao and M. Georgeff, BDI agents: From theory to practice, in Proceedings of the *First International Conference on Multiagent Systems*, San Francisco June 12-14, 1995.
- [33] A. S. Rao, AgentSpeak(L): BDI agents speak out in a logical computable language, in Proceedings of the *7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away*, Eindhoven, The Netherlands, 1996, pp. 42-55.
- [34] J. A. Rice, *Mathematical Statistics and Data Analysis*, Duxbury Press, 1999.
- [35] A. Suna and A. E. F. Seghrouchni, Programming Mobile Intelligent Agents: an Operational Semantics, *Web Intelligence and Agent Systems*, IOS Press, **5** (1) (2007). 47-67.
- [36] R. Vieira, A. F. Moreira, M. Wooldridge and R. H. Bordini, On the Formal Semantics of Speech-Act Based Communication in an Agent-Oriented Programming Language, *Journal of Artificial Intelligence Research (JAIR)*, **29** (2007). 221-267.
- [37] M. Wooldridge and N. R. Jennings, Agent Theories, Architectures and Languages: A Survey, in M. Wooldridge and N. R. Jennings, eds., *Intelligent Agents - Theories, Architectures, and Languages I*, Springer-Verlag, 1995.
- [38] M. Wooldridge, *Reasoning about rational agents*, The M. I. T. Press, Cambridge, MA, 2000.
- [39] M. Wooldridge, *An Introduction to Multiagent Systems*, John Wiley & Sons (Chichester, England), 2002.
- [40] H. Zhang and S. Y. Huang, A Parallel BDI Agent Architecture, in Proceedings of the *2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2005)*, Compiègne University of Technology, France, September 19-22, 2005, pp. 157-160.
- [41] H. Zhang and S. Y. Huang, BDIE architecture for rational agents, in Proceedings of the *international conference for Integration of Knowledge Intensive Multi-Agent Systems (KIMAS05)*, Waltham, Massachusetts, US, April 18-21, 2005, pp. 623-628.
- [42] H. Zhang and S. Y. Huang, Are Parallel BDI Agents Really Better?, in Proceedings of the *17th European Conference on Artificial Intelligence*, Riva del Garda, Italy, August 28-September 1, 2006.
- [43] H. Zhang and S. Y. Huang, A General Framework for Parallel BDI Agents, in Proceedings of the *2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2006)*, Hong Kong, China, December 18-22, 2006.