Handling Emergency Goals in HTN Planning

Hisashi Hayashi *† Seiji Tokura *† Fumio Ozaki *† Tetsuo Hasegawa *‡

Abstract— **Integration of deliberation and reaction has been an important research topic concerning agents in view of the need for an agent to react tentatively and immediately to the changing world when unexpected events occur while executing a plan. An agent is not supposed to think for a long time before reacting. Also, its reaction is not supposed to change the world greatly. However, there are some cases where deliberation is necessary for achieving an emergency goal or where the emergency plan execution prevents the resumption of the suspended plan execution. This paper presents a new concept of on-line interruption planning that integrates deliberation and emergency deliberation. When an emergency goal is given while executing a plan, our agents suspend the current plan execution, make and execute an emergency plan, and resume the suspended plan execution. Because our agents continuously modify the suspended plans while executing an emergency plan, they can resume the suspended plans correctly and efficiently even if the world has changed greatly due to the emergency plan execution.**

Keywords: planning, agent, robotics

1 Introduction

Handling asynchronous "emergency" goals is a very important subject of research in planning. Asynchronous goals are inputted to the planning agent even while executing a plan for another goal. The simplest way of handling asynchronous goals is to plan and execute goals in a first-come-first-served manner. However, in this approach, even when receiving an emergency goal, it will be served last.

A better approach for handling asynchronous emergency goals is to prioritize multiple goals where the goal with the highest priority is served first. Intention scheduling in BDI (Belief, Desire, Intention) agents is researched in [33]. Here, intentions are committed plans for different goals. Priorities and interferences between goals are taken into consideration. Most current BDI agents [3, 7, 18, 21, 24] are based on PRS [14]. However, the problem to use PRS-like BDI agents is that they can neither plan nor replan. Instead, they reactively select and commit to ready-made plans.

There is a planning agent called ROGUE [15] that can make plans for multiple asynchronous goals with priorities. ROGUE uses an on-line partial-order backward-chaining planner called PRODIGY. When ROGUE receives a new goal while executing a plan, it adds the new goal to the search space of the current plan. Then, PRODIGY incrementally expands the plan for the new goal without invalidating the current plan. The execution order is calculated based on the priorities of goals. It seems that this approach is promising. However, in the case of emergency, the agent should suspend the current plan execution, execute the emergency plan immediately, and modify the suspended plan accordingly. When planning for the emergency goal, it is not necessary to keep the suspended plan valid as in PRODIGY. Although we share a similar motivation with ROGUE and PRODIGY, we would like to execute the best plan (in terms of costs) for the emergency goal rather than an incrementally expanded plan for multiple asynchronous goals. Therefore, the motivation is rather different.

In this paper, we will present the new concept of "interruption planning" for asynchronous emergency goals. Here, interruption planning means that when the planning agent receives an emergency goal, it suspends the current plan execution, makes emergency plans, executes an emergency plan immediately, and resumes the suspended plan execution. In our new interruption planning, while executing a plan for the emergency goal, the agent keeps and continuously updates the emergency plans and the suspended initial plans. This means that our interruption planning is on-line planning. Unlike ROGUE, in our interruption planning, we make the best plan (in terms of cost) for the emergency goal without trying to reuse any parts of plans for the suspended goals. Also, in most planning systems including PRODIGY, actions are treated as the primitive tasks that cannot be suspended. In our new interruption planning, in order to handle an emergency goal immediately, the planning agent tries to suspend the current action execution if it does not contribute to the achievement of the emergency goal. This is especially important if it takes time to execute an action. We will also show that this action suspension is effective for realizing interruption planning in stratified multi-agent systems. In stratified multi-agent planning, the parent agent makes and executes rough plans, and the child agent makes and executes detailed plans. An

*Toshiba Corporation, 1 Komukai-Toshiba-cho, Saiwai-ku, Kawasaki, 212-8582 Japan. Email: {hisashi3.hayashi, seiji.tokura, fumio.ozaki, tetsuo3.hasegawa}@toshiba.co.jp
†Corporate Research and Development Center
‡Corporate Software Engineering Center

action of the parent agent corresponds to a whole plan of the child agent. This means that the action execution time of the parent agent is generally long and changes the world greatly. Therefore, it is very important to suspend unnecessary action execution of the parent agent.

2 Museum Guide Scenario

In this section, we introduce a museum guide scenario as an example to illustrate interruption planning. Subsequently, this scenario will also be used for experimental evaluation. Figure 1 shows the map of a museum where the robot moves. Nodes are places where the robot localizes itself relative to the map with the help of, for example, markers which can be recognized through image processing. Especially, nodes are set at intersections of paths or points of interests. The robot moves from one node to the next node along an arc. When the user specifies the destination (node), the robot takes the person there.

The museum is divided into some areas. Given the destination, the robot first searches a rough route that connects only areas. Then the robot searches a detailed route in the first area that connects nodes. For example, when moving from n1 (area1) to n40 (area8), the robot first makes a rough plan: $area1 \rightarrow area3 \rightarrow area5 \rightarrow area7 \rightarrow area8$. The robot then thinks about how it should move to the next area: $n1 \rightarrow n6 \rightarrow n10$.

While taking a person to a node, suppose that the robot is told to go to a toilet. This is an emergency goal and the user cannot wait until the tour guide ends. Therefore, the robot should suspend the current plan execution, take the person to the toilet node, and resume the tour guide.

Note that we use this scenario to evaluate not the efficiency of route planning but the efficiency of on-line interruption planning.

3 On-line Planning in Dynagent

In order to implement interruption planning, we will use an on-line forward-chaining HTN planning agent called Dynagent [16] in our pilot implementation. Here, an on-line planning agent means an agent that interleaves planning, belief updates, and execution. When new information is found and the belief is updated unexpectedly the on-line planning agent modifies its plans even while executing a plan.

HTN planning [6, 22, 26, 28, 32] is different from standard planning which just connect the "preconditions" and "effects" of actions. It makes plans, instead, by decomposing abstract tasks into more concrete subtasks or subplans, which is similar to Prolog that decomposes goals into subgoals. Forward-chaining HTN planning [16, 22] is especially suitable for a dynamically changing world because some task decompositions can be suspended when planning initially and resumed, using the most recent knowledge, just before the abstract tasks are executed. Other merits of HTN planning, such as efficiency, expressiveness of domain knowledge and planning control knowledge are discussed in [9, 22, 30].

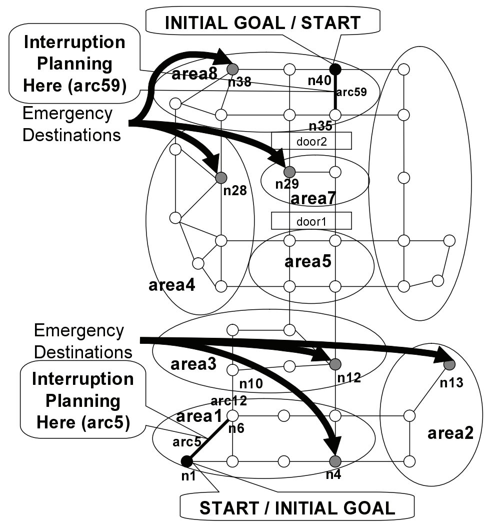

Figure 1: The Map

Dynagent keeps several alternative plans and incrementally modifies the alternative plans while executing a plan. These alternative plans can contain abstract tasks but only the first task of each alternative plan has to be an executable action. As shown in Figure 5, in order to implement on-line interruption planning based on Dynagent, the agent has to maintain not only the plans for the emergency goals but also the plans for the suspended initial goal. Dynagent estimates the cost of each plan using A*-like heuristics and searches the best plan in terms of costs.

4 When Suspending an Action

When an emergency goal is given, the agent should suspend the current plan execution. If the time for action execution is short, for the planning agent to wait till the action executor finishes the action execution poses no problem. However, the action execution time is generally long in such areas as robotics, and we would like to suspend the current action execution immediately and start the execution of the emergency plan. Therefore, as Figure 2 shows, before executing a plan for an emergency goal, the planning agent asks if the action executor can suspend the current action execution. If it is possible, the planning agent tells the action executor to stop the current action execution. After the action suspension, the planning agent updates its belief. For example, if the planning agent suspends the action to go from $n1$ to $n6$ along $arc5$, then the location of the robot will be on $arc5$.

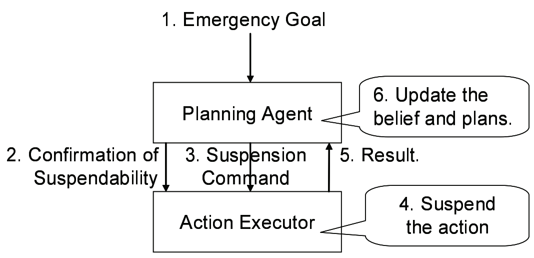

Figure 2: Action Suspension

We assume the planning agent knows the effect of action suspension.

If the execution of an emergency plan does not affect the suspended plan, then the agent can resume the initial plan execution after resuming the suspended action. If the suspended plan is invalidated by the effect of action suspension, then the plan has to be changed to an alternative plan, in which case we need to rollback the suspended action if necessary.

Figure 3 shows how to replace the suspended action in a plan with the “resuming action.” We assume that the planning agent knows the precondition and effect of the resuming action. For example, if the planning agent suspends the action to go to $n6$ along $arc5$, then the precondition of the resuming action is that the location of the robot is on $arc5$, and the effect is that the location of the robot becomes $n6$. The resuming action might have the effect that invalidates the rest of the plan. Also the precondition of the resuming action needs to be checked. Therefore, we need to recheck the satisfiability of the preconditions of actions in the modified plan.

On the other hand, the “rollback action” should be added to each alternative plan, as shown in Figure 4, if the first action is different from the suspended action. We assume that the planning agent knows the precondition and effect of the rollback action. For example, if the agent suspends the action to go to $n6$ along $arc5$, then the precondition of the rollback action is that the location of the robot is on $arc5$, and the effect is that the location of the robot becomes $n1$. The rollback action might have an effect that invalidates the rest of the plan. Also the precondition of the rollback action needs to be checked. Therefore, we need to recheck the satisfiability of the preconditions of actions in the modified plan.

Considering the effects of action suspension, the resuming actions, and rollback actions, we define the following algorithm for the modifications of the belief and plans when suspending an action. In the following algorithm, we assume that there exists the resuming action for each suspendable action. We do not care even if rollback actions do not exist, in which case we do not add the rollback actions to alternative plans.

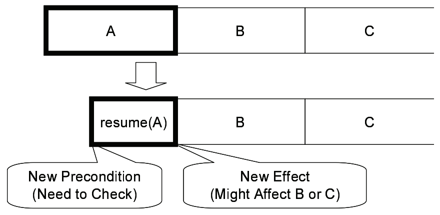
A
B
C
resume(A)
B
C
New Precondition (Need to Check)
New Effect (Might Affect B or C)

Figure 3: Replacing with the Resuming Action

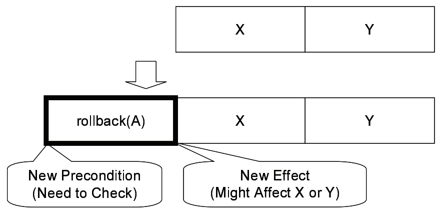
X
Y
rollback(A)
X
Y
New Precondition (Need to Check)
New Effect (Might Affect X or Y)

Figure 4: Adding the rollback Action

Algorithm 1 *(Modifications of Belief B and Plans PS when Suspending Action A.)*

1. *(Belief Update) Update the current belief B based on the effect of suspension of the action A.*
2. *(New Plan Creation) If new valid plans can be created, then add the new valid plans to PS. (To find new valid plans, we use the algorithm of Dynagent [16].)*
3. *(Plan Modification) For each plan P in PS, modify P as follows:*
   (a) *(Adding Resuming Actions) If the first action of the plan P is identical to A, then replace the occurrence of A in P with the resuming action $resume(A)$ of A.*
   (b) *(Adding rollback Actions) If the first action of the plan P is not identical to the suspended action A, and if there exists the rollback action $rollback(A)$ of A, then add $rollback(A)$ to the top of the plan.*
   (c) *(Removing Invalid Plans) Based on the current belief B, if a precondition of an action in P is not satisfied, then remove P from PS.*

5 Agent Algorithm

In this section, we define the agent algorithm for on-line interruption planning. Two types of goals are given to the planning agent: normal goals and emergency goals. Normal goals are kept in a waiting goal list and the planning agent handles them sequentially as usual. On the other hand, when the planning agent receives an emergency goal, the current plan execution is interrupted as

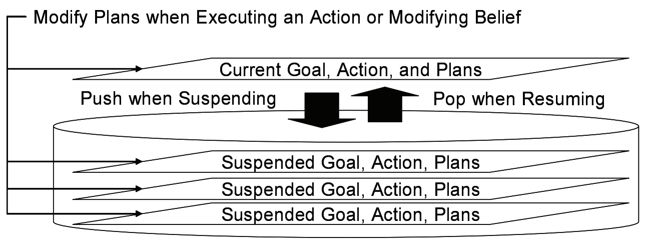
Modify Plans when Executing an Action or Modifying Belief
Current Goal, Action, and Plans
Push when Suspending
Pop when Resuming
Suspended Goal, Action, Plans
Suspended Goal, Action, Plans
Suspended Goal, Action, Plans

Figure 5: Stack for Suspended Goals

soon as possible to handle the emergency goal. Normal goals are handled by Algorithm 4. Emergency goals are handled by Algorithm 3. The algorithm for planning and plan execution is defined in Algorithm 2. Although we definitely need to handle normal goals by Algorithm 4, the originality can be found only in Algorithm 3 and in Algorithm 2.

As Figure 5 shows, the agent keeps the suspended goal, the suspended plans, and the suspended action in "the stack for suspended goals." While executing an "emergency plan," the agent continuously updates not only the plans for the emergency goal but also the plans in the stack. To modify these plans, when updating the belief or executing an action, we can use the algorithm of Dynagent [16].

Also, in order to resume the suspended action, the action executor might keep some information while handling an emergency goal. When the action resumption becomes no longer necessary, because of the change of the plan, then the planning agent tells the action executor to clear the recorded state for the action resumption.

Now we define an algorithm for planning and plan execution. This algorithm is started from Algorithm 3, Algorithm 4, or Algorithm 5 and is defined below. This algorithm is also used when resuming the suspended plan execution. In this case, the suspended plans and the suspended action are recorded in association with the given goal. Note that when another thread (Algorithm 3 or Algorithm 5) makes the status of the goal "suspended," the following algorithm is finished.

Algorithm 2 *(Planning and Plan Execution)*

1. *(Goal Input) A goal is given as an input.*

2. *(Input of Suspended Plans/Action) If the status of the given goal is "suspended," then the suspended plans and the suspended action (if it exists) are also given as inputs.*

3. *(Planning) Make the plans for the goal. (Use the HTN planning algorithm of Dynagent [16].)*

4. *(Plan Selection) Select a plan to execute from the alternative plans.*

5. *(Clearance of the Suspended Action) If the status of the goal is "suspended," there exists a suspended action, and the suspended action is different from the next action of the selected plan, then tell the action executor to abandon the recorded state for the action resumption.*

6. *Set the status of the goal to "active."*

7. *(Plan Execution Loop) Repeat the following procedure while the status of the goal is "active:"*

    (a) *(Action Execution) Following the selected plan, tell the action executor to execute the next action and wait for the result ("success", "failure", or "suspended") that is reported from it.*

    (b) *(Update of the Belief and Plans) If the result of the action execution is either "success" or "failure," then modify the belief and all the plans, including the plans recorded in the stack for suspended goals, following the plan modification algorithm[1] of Dynagent [16].*

    (c) *(Update of the Belief and Plans) If the result of the action execution is "suspended," then modify the belief and all the plans, including the plans recorded in the stack for suspended goals, following Algorithm 1.*

    (d) *(Successful Plan Execution) If one of the plans is successful, then change the status of the goal to "success."*

    (e) *(Plan Execution Failure) If no alternative plan exists, then change the status of the goal to "failure."*

    (f) *(Plan Selection) If the status of the goal is "active," then select a plan from alternative plans.*

8. *Output the status of the goal ("success", "failure", or "suspended.")*

9. *If the status of the goal is "suspended," then output the suspended plans and the suspended action (if it exists.)*

[1]The plan modification of Dynagent includes deletion of invalid plans, addition of new valid plans, and deletion of the executed action.

The following algorithm shows how to process emergency goals. When an emergency goal is given, the planning agent tries to suspend the current plan execution to execute a plan for the emergency goal as soon as possible. The suspended plan will be resumed after the emergency plan execution. Note that when the following algorithm changes the status of the goal to "suspended," the plan execution process (Algorithm 2) is finished.

Algorithm 3 *(Emergency Goal Handling)*

1. *(Goal Input) An emergency goal A is given as an input.*
2. *(Normal Plan Execution) If there does not exist a goal whose plan is being executed by Algorithm 2, then start the plan execution process (Algorithm 2) for A.*
3. *If there exists a goal B whose plan is being executed, then execute the following procedure:*
    - (a) *Change the status of B to "suspended."*
    - (b) *(Action Suspension) If an action is being executed, ask the action executor if it is possible to suspend the action that is being executed. If the action can be suspended, then tell the action executor to suspend the action execution.*
    - (c) *(Plan Suspension) Wait till the plan execution process (Algorithm 2) for B is finished and receive the suspended plans and the suspended action (if it exists.)*
    - (d) *(Pushing to Stack) Push the set of the suspended goal B, the suspended plans, and the suspended action (if it exists) to the stack for suspended goals.*
    - (e) *(Emergency Plan Execution) Start the goal handling process (Algorithm 2) for A and wait for the result.*
    - (f) *(Popping from Stack) Pop the set of the suspended goal B, the suspended plans, and the suspended action (if it exists) from the stack for suspended goals.*
    - (g) *(Plan Resumption) Restart the plan execution algorithm (Algorithm 2) for B, inputting the suspended plans and the suspended action (if it exists) with B.*

The following algorithm shows how to process normal goals. As explained before, normal goals are kept in a waiting goal list and the planning agent handles them sequentially as usual. This algorithm is necessary. However, in terms of originality, it is not important.

Algorithm 4 *(Normal Goal Handling)*

1. *A normal goal A is given as an input.*
2. *Add A to the waiting goal list as the last element.*
3. *Wait till A becomes the first element of the waiting goal list and no goal is being processed by Algorithm 2, and the stack for suspended goals becomes empty.*
4. *Remove A from the waiting goal list.*
5. *Start the plan execution process (Algorithm 2) for A.*

6 Stratified Multi-agent Interruption Planning

In stratified multi-agent planning systems, the parent planning agent executes a rough plan by giving subgoals (= actions of the parent planning agent) to its child planning agents. Given a subgoal, the child planning agents make and execute a detailed plan. For the parent planning agent, the child planning agents are just action executors, and the parent planning agent does not know how its action executors or child planning agents are implemented. The child planning agent can handle the subgoal using the algorithm defined in the previous section. On-line planning in a stratified multi-agent system is researched in [17]. However, interruption planning has not been incorporated into stratified multi-agent systems.

In stratified multi-agent systems, an action of the parent agent corresponds to a whole plan of the child planning agent. Therefore, it is important to suspend meaningless action execution. As shown in Figure 6 (compare with Figure 2), a plan suspension instruction from the parent planning agent causes another action suspension of the child planning agent. Similarly, a plan resumption (or clearance) instruction causes another action suspension (respectively, clearance) of the child planning agent. The following algorithm shows how this can be done. In the same way, it is possible to use a grandchild planning agent which is a child of the child planning agent. It is also

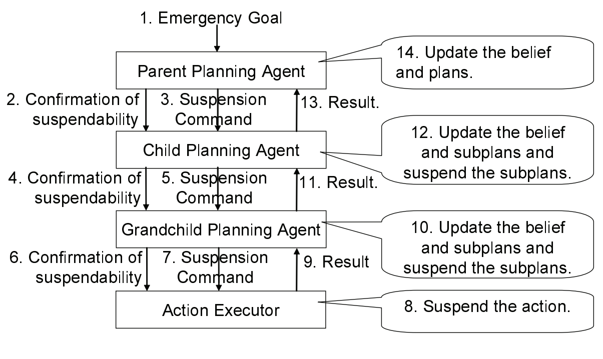
1. Emergency Goal
Parent Planning Agent
14. Update the belief and plans.
2. Confirmation of suspendability
3. Suspension Command
13. Result.
Child Planning Agent
12. Update the belief and subplans and suspend the subplans.
4. Confirmation of suspendability
5. Suspension Command
11. Result.
Grandchild Planning Agent
10. Update the belief and subplans and suspend the subplans.
6. Confirmation of suspendability
7. Suspension Command
9. Result
Action Executor
8. Suspend the action.

Figure 6: Subplan Suspension

possible for the parent planning agent to have more than two child planning agents. However, because Dynagent does not execute actions in parallel, two child planning agents do not work at the same time. Note that when the following algorithm changes the status of the goal to "suspended," the plan execution process (Algorithm 2) is finished.

Algorithm 5 *(Subplan Suspension)*

1. *(Instruction Input) A suspension instruction of the current goal G is given as an input (from the parent planning agent.)*
2. *Change the status of G to "suspended."*
3. *(Action Suspension) If an action is being executed, ask the action executor if it is possible to suspend the action that is being executed. If the action can be suspended, then tell the action executor to suspend the action execution.*
4. *(Plan Suspension) Wait till the plan execution process (Algorithm 2) for G is finished and receive the suspended plans and the suspended action (if it exists.)*
5. *(Pushing to Stack) Push the set of the suspended goal G, the suspended plans, and the suspended action (if it exists) to the stack for suspended goals.*
6. *(Instruction Waiting) Wait for the resumption or state clearance instruction of G from the parent planning agent.*
7. *(Popping from Stack) Pop the set of the suspended goal G, the suspended plans, and the suspended action (if it exists) from the stack for suspended goals.*
8. *(State Clearance) When receiving the state clearance instruction of G, abandon the suspended goal, the suspended plans, and the suspended action and tell the action executor to abandon the recorded state for the action resumption.*
9. *(Plan Resumption) When receiving the resumption instruction of G, restart the plan execution algorithm (Algorithm 2) for G, inputting the suspended plans and the suspended action (if it exists) with G.*

7 Experiments

This section evaluates the efficiency of replanning when resuming the suspended plans by means of experiments based on the museum guide scenario explained in Section "Museum Guide Scenario". However, it is not the aim of the experiments to measure the efficiency of route planning. Our planning algorithm can be used for other purposes.

We use two stratified planning agents. The parent planning agent is in charge of the area movement planning and makes a plan to move from one area to another. (See Figure 1.) The parent planning agent tells the child planning agent to execute the action to move to the next area or a node inside an area. The child planning agent is in charge of the node movement inside an area and makes a plan to move from one node to another. When moving to the next area, the child planning agent also updates the area map in the memory, following the instructions from the parent planning agent. For the planning agent, the child planning agent is an action executor. We assume that the doors are always open. However, these agents can dynamically change the plans, as shown in [17], if a door on the route is closed during the plan execution, but that is not what we wish to show in this paper.

Initially the robot is at $n40$ in $area8$. We give the goal to go to $n1$ in $area1$ to the parent planning agent. While the robot is moving from $n40$ in $area8$ to $n35$ along $arc59$, we give an emergency goal to go to another place ($n38$ in $area8$ or $n28$ in $area4$ or $n29$ in $area7$). After executing an emergency plan and visiting the node, the parent planning agent resumes the initial plan to go to $n1$ in $area1$. We measure this replanning time for the plan resumption. We compare the naive replanning method to plan from scratch and our on-line replanning method. We measure this replanning time for the plan resumption. Ideally, we would like to compare our on-line interruption planning technique with other on-line interruption planning techniques. However, because the concept of interruption planning is new, our agent is the only on-line interruption planning agent.

We conducted similar experiments. Initially the robot is at $n1$ in $area1$. We give the goal to go to $n40$ in $area8$ to the parent planning agent. While the robot is moving from $n1$ in $area1$ to $n6$ along $arc5$, we give an emergency goal to go to another place ($n13$ in $area2$ or $n4$ in $area1$ or $n12$ in $area3$). After executing an emergency plan and visiting the node, the parent planning agent resumes the initial plan to go to $n40$ in $area8$. We measure this replanning time for the plan resumption.

We compare our on-line replanning method with the naive replanning method to plan from scratch. Each experiment was conducted three times and the average time is shown in Table 1, Table 2, and Table 3. The agent system is implemented in Java and the planner that the planning agents use is implemented in Prolog and Java. We used a PC (Windows XP) equipped with a Pentium4 2.8GHz and 512MB of RAM.

Table 1 shows the replanning time of the parent planning agent. When the parent planning agent does not need to correct the plan, our on-line replanning method is around 30 times as efficient as the naive replanning method. For example, while the robot is moving in $area8$ to go to *area*1, even if the emergency goal to go to *area*7 is given, the initial plan of the parent planning agent is not affected because the emergency destination *area*7 is on the route from *area*8 to *area*1. In the other cases, our on-line replanning method is still more than twice as fast as the naive replanning method.

Table 2 shows the replanning time of the child planning agent. When the emergency goal is given, if the emergency destination and the current location of the robot are in the same area, then our on-line replanning method is 2 - 3 times as efficient as the naive replanning method. In the other cases, the efficiency of our on-line replanning method is almost the same as the efficiency of the naive replanning method. This is because our planning agents make plans from scratch when the goal is changed.

Table 3 shows the total replanning time of the parent planning agent and the child planning agent. Our on-line replanning method is twice as efficient as the naive replanning method when the parent planning agent needs to modify the plan. When the parent planning agent does not need to correct the plan, our on-line replanning method is much faster than the naive replanning approach.

Table 1: Parent Agent Replanning Time

Starting Point	Initial Destination	Place of Interruption	Emergency Destination	On-line Replanning	Naive Replanning
n40(area8)	n1(area1)	arc59(area8)	n38(area8)	0.01 sec	0.37 sec
n40(area8)	n1(area1)	arc59(area8)	n28(area4)	0.13 sec	0.30 sec
n40(area8)	n1(area1)	arc59(area8)	n29(area7)	0.01 sec	0.30 sec
n1(area1)	n40(area8)	arc5(area1)	n13(area2)	0.24 sec	0.50 sec
n1(area1)	n40(area8)	arc5(area1)	n4(area1)	0.01 sec	0.36 sec
n1(area1)	n40(area8)	arc5(area1)	n12(area3)	0.01 sec	0.31 sec

Table 2: Child Agent Replanning Time

Starting Point	Initial Destination	Place of Interruption	Emergency Destination	On-line Replanning	Naive Replanning
n40(area8)	n1(area1)	arc59(area8)	n38(area8)	0.06 sec	0.15 sec
n40(area8)	n1(area1)	arc59(area8)	n28(area4)	0.09 sec	0.11 sec
n40(area8)	n1(area1)	arc59(area8)	n29(area7)	0.05 sec	0.05 sec
n1(area1)	n40(area8)	arc5(area1)	n13(area2)	0.05 sec	0.06 sec
n1(area1)	n40(area8)	arc5(area1)	n4(area1)	0.03 sec	0.08 sec
n1(area1)	n40(area8)	arc5(area1)	n12(area3)	0.06 sec	0.07 sec

Table 3: Total Replanning Time

Starting Point	Initial Destination	Place of Interruption	Emergency Destination	On-line Replanning	Naive Replanning
n40(area8)	n1(area1)	arc59(area8)	n38(area8)	0.06 sec	0.52 sec
n40(area8)	n1(area1)	arc59(area8)	n28(area4)	0.22 sec	0.41 sec
n40(area8)	n1(area1)	arc59(area8)	n29(area7)	0.05 sec	0.35 sec
n1(area1)	n40(area8)	arc5(area1)	n13(area2)	0.29 sec	0.56 sec
n1(area1)	n40(area8)	arc5(area1)	n4(area1)	0.04 sec	0.44 sec
n1(area1)	n40(area8)	arc5(area1)	n12(area3)	0.07 sec	0.38 sec

8 Related Work

8.1 Integration of Deliberation and Reaction

Integration of deliberation and reaction has been an important subject of research to realize autonomous agents working in dynamic environments. While executing a plan for a goal, even the deliberative agent needs to react to an unexpected event or situation in the case of an emergency. Normally, when combining plan execution and reaction, the agent does not plan to react, and the reaction does not change the world greatly. On the other hand, in our approach, the agent does plan for an emergency goal, and the emergency plan might change the world greatly.

One way [19] of combining plan execution and reaction is to repeat the following cycle: 1. observe; 2. react if necessary; 3. update the plan; 4. act following the plan; 5. go to 1. It seems that the agent can react to the changing world while executing a plan. However, this algorithm does not take into account the time for action execution. If it takes time to act, the agent cannot react quickly as long as we use this algorithm.

Another way of solving this problem is to use the idea of layered architecture [4] where the (higher-level) deliberative planning agent controls the (lower-level) action executor. Here, the planning agent and the action executor are working concurrently. Therefore, the action executor reacts immediately to the world without considering the plan execution. Unless the reaction causes action failure, the action executor does not have to report the reaction to the planning agent. In the case of action failure, the action executor reports it to the planning agent, and the planning agent replans.

8.2 On-line Planning

Our interruption planning is an extension of on-line planning that interleaves planning and execution. While executing a plan, the world might change unexpectedly. By selecting and executing one plan, some alternative plans might become invalid. In interruption planning, suspended plans might be affected by the execution of an emergency plan. In that case, on-line planners modify the plans incrementally. In order to detect an error in a plan, it is necessary to confirm that each action precondition in the plan is not affected. For this purpose, PLANEX [13] uses triangle tables. (PLANEX is the execution monitoring system of the well-known classical planner STRIPS [12].) Also, causal links of partial-order plans which were introduced first in NONLIN [28] can be used to detect an error in a plan. Causal links are used in many backward-chaining partial-order planners such as TWEAK [5] and SNLP [20]. As explained in [25], IPEM [1] is the first on-line planner which smoothly integrates partial-order backward-chaining planning and execution. PRODIGY which was introduced in Introduction is also an on-line partial-order backward-chaining planner. SIPE [32] is known as an on-line HTN planner. The plan repairing strategy based on O-Plan [6] and NONLIN [28], both of which are partial-order HTN planners, is explained in [10]. Dynagent [16], which we used in this paper, is an on-line forward-chaining HTN planner that extends the algorithm of SHOP [22].

8.3 Integration of BDI and Planning

PRS-like BDI agents reactively commit to ready-made plans. However, there are some agent systems that integrate PRS-like BDI agent models and planning. CANPLAN [27] combines PRS-like BDI agent models and HTN planning based on the similarities of their languages. However, replanning is still future work. Another BDI agent system [29] calls the planner when precompiled plans are hard to devise in advance. However, planning under uncertainty is the future work. Cypress [31] combines a BDI system (PRS-CL) and an on-line HTN planner (SIPE-2). Interestingly, even when PRS-CL detects an error during plan execution, it continues executing unaffected parts of the plan while SIPE-2 is producing an alternative plan.

8.4 Multi-agent Planning

As surveyed in [8, 9, 11], there are mainly two kinds of multi-agent planning: 1. Planning for task distribution to agents; 2. Coordination of plans of different agents. Typically, when planning for task distribution, the parent agent makes a rough plan and distributes tasks to its child agents following the plan. Stratified multi-agent planning is related to planning for task distribution. On-line planning in stratified multi-agent systems is researched in [17]. Coordination of plans is necessary because each agent has a different goal and plan execution of one agent affects plan execution of another agent. Dynamic recoordination of plans is researched in [2]. A good survey on distributed and continual (= on-line) planning can be found in [9].

9 Conclusions

We have introduced the new concept of interruption planning and shown how interruption planning can be implemented. When receiving an emergency goal, the agent suspends the current plan execution, generates and executes the emergency plan as soon as possible, and modifies the suspended plans accordingly. As explained in Introduction, this is different from incremental plan expansion of PRODIGY which tries to reuse the current plan when receiving an asynchronous goal with a priority. We have also shown how interruption planning can be implemented in stratified multi-agent systems. As soon as the planning agent receives an emergency goal, it tries to interrupt unnecessary action execution. In the same way, as soon as the planning agent receives an emergency goal, it tries to interrupt unnecessary plan execution of its child planning agent. Our interruption planning is a kind of on-line planning, and the planning agent keeps and continuously updates the suspended plans. Therefore, when resuming the plan execution, the agent replans as quickly as our experiments have shown.

The new concept of interruption planning also combines deliberation and reaction because when the agents quickly make and execute a short emergency plan, the plan can be regarded as a reaction. Unlike previous approaches to combine deliberation and reaction, where reaction is not produced by deliberation, our interruption planning combines deliberation and emergency deliberation.

In order to implement interruption planning, we have extended the on-line forward-chaining HTN planning algorithm of Dynagent. However, we think, it is possible to use and extend on-line planning algorithms of classical planning. The comparison of HTN planning and classical planning in terms of interruption planning is our future work.

It is interesting to use our interruption planning in many application areas [23] of forward-chaining HTN planning. Currently, we are trying to apply our interruption planning to a real museum guide robot which takes a guest to destinations (POIs). During the museum tour, the guest might ask the robot to take him/her to a toilet. While explaining an exhibit, if the guest asks a question, then the robot has to suspend the explanation, answer the question, and resume the explanation. The robot might need to say "Hello!" immediately when it recognizes a person during the museum guide tour. So far, we have successfully tested these scenarios. Especially, the first scenario was tested using a real mobile robot called ApriAttenda [34]. However, this experiment was conducted inside a small laboratory. In the future, we would like to run this robot in a real museum environment.

References

[1] Ambros-Ingerson, J., and Steel, S. 1988. Integrating planning, execution and monitoring. In *AAAI88*, 735–740.

[2] Bartold, T., and Durfee, E. 2003. Limiting disruption in multiagent replanning. In *AAMAS03*, 49–56.

[3] Braubach, L.; Pokahr, A.; and Lamersdorf, W. 2005. Jadex: A BDI-agent system combining middleware and reasoning. In *Software Agent-Based Applications, Platforms and Development Kits*. Birkhäuser Book. 143–168.

[4] Brooks, R. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* 2(1):14–23.

[5] Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

[6] Currie, K., and Tate, A. 1991. O-plan: The open planning architecture. *Artificial Intelligence* 52(1):49–86.

[7] Dastani, M.; van Riemsdijk, B.; Dignum, F.; and Meyer, J.-J. 2003. A programming language for cognitive agents: Goal directed 3APL. In *ProMAS03*, 111–130.

[8] de Weerdt, M.; ter Mors, A.; and Witteveen, C. 2005. Multi-agent planning: An introduction to planning and coordination. In *Handouts of the European Agent Summer School*, 1–32.

[9] desJardins, M.; Durfee, E.; Ortiz, C.; and Wolverton, M. 1999. A survey of research in distributed, continual planning. *AI Magazine* 20(4):13–22.

[10] Drabble, B.; Tate, A.; and Dalton, J. 1997. Repairing plans on-the-fly. In *NASA Workshop on Planning and Scheduling for Space*.

[11] Ferber, J. 1999. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley.

[12] Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

[13] Fikes, R.; Hart, P.; and Nilsson, N. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3(4):251–288.

[14] Georgeff, M., and Ingrand, F. 1989. Decision-making in an embedded reasoning system. In *IJCAI89*, 972–978.

[15] Haigh, K., and Veloso, M. 1998. Interleaving planning and robot execution for asynchronous user requests. *Autonomous Robots* 5(1):79–95.

[16] Hayashi, H.; Tokura, S.; Hasegawa, T.; and Ozaki, F. 2006. Dynagent: An incremental forward-chaining HTN planning agent in dynamic domains. In *Post-Proceedings of DALT05*, LNAI 3904, 171–187. Springer.

[17] Hayashi, H. 2007. Stratified multi-agent HTN planning in dynamic environments. In *KES AMSTA07*, 189–198.

[18] Howden, N.; Rőnnquist, R.; Hodgson, A.; and Lucas, A. 2001. JACK: Intelligent agents - summary of an agent infrastructure. In *IAMSMAS01*.

[19] Kowalski, R., and Sadri, F. 1999. From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence* 25:391–419.

[20] McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *AAAI91*, 634–639.

[21] Morley, D., and Myers, K. 2004. The SPARK agent framework. In *AAMAS04*, 712–719.

[22] Nau, D.; Cao, Y.; Lotem, A.; and Mũnoz-Avila, H. 1999. SHOP: simple hierarchical ordered planner. In *IJCAI99*, 968–975.

[23] Nau, D.; AU, T.-C.; Ilghami, O.; Kuter, U.; Wu, D.; and Yaman, F. 2005. Applications of SHOP and SHOP2. *IEEE Intelligent Systems* 20(2):34–41.

[24] Rao, A. 1996. AgentSpeak(L): BDI agents speak out in a logical computable language. In *MAAMAW96*, 42–55.

[25] Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

[26] Sacerdoti, E. 1977. *A Structure for Plans and Behavior*. American Elsevier.

[27] Sardina, S.; de Silva, L.; and Padgham, L. 2006. Hierarchical planning in BDI agent programming languages: A formal approach. In *AAMAS06*, 1001–1008.

[28] Tate, A. 1977. Generating project networks. In *IJCAI77*, 888–893.

[29] Walczak, A.; Braubach, L.; Pokahr, A.; and Lamersdorf, W. 2006. Augmenting BDI agents with deliberative planning techniques. In *ProMAS06*.

[30] Wilkins, D., and desJardins, M. 2001. A call for knowledge-based planning. *AI Magazine* 22(1):99–115.

[31] Wilkins, D.; Myers, K.; Lowrance, J.; and Wesley, L. 1995. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI* 7(1):197–227.

[32] Wilkins, D. 1988. *Practical Planning*. Morgan Kaufmann.

[33] Yan, S.-B.; Lin, Z.-N.; Hsu, H.-J.; and Wang, F.-J. 2005. Intention scheduling for BDI agent systems. In *COMPSAC05*, 133–140.

[34] Yoshimi, T.; Nishiyama, M.; Sonoura, T.; Nakamoto, H.; Tokura, S.; Sato, H.; Ozaki, F.; Matsuhira, N.; and Mizoguchi, H. 2006. Development of a person following robot with vision based target detection. In *IROS06*, 5286–5291.