Active Execution Monitoring Using Planning and Semantic Knowledge

Abdelbaki Bouguerra and **Lars Karlsson** and **Alessandro Saffiotti***
Örebro University, SE-70182 Örebro Sweden
Email: {aba,lkn,asaffio}@aass.oru.se
Web: http://aass.oru.se

Abstract

To cope with the dynamics and uncertainty inherent in real world environments, autonomous mobile robots need to perform execution monitoring for verifying that their plans are executed as expected. Domain semantic knowledge has lately been proposed as a source of information to derive and monitor implicit expectations of executing actions. For instance, when the robot moves into an office, it would expect to see a desk and a chair. Such expectations are checked using the immediately available perceptual information. We propose to extend the semantic knowledge-based execution monitoring to handle situations where some of the required information is missing. To this end, we use AI sensor-based planning to actively search for such information. We show how verifying execution expectations can be formulated and solved as a planning problem involving sensing actions. Our approach is illustrated by showing test scenarios run in an indoor environment using a mobile robot.

Introduction

Plan execution and monitoring by mobile robots is a complex task as it involves dealing with uncertainty and unexpected situations. To deal with potential run-time contingencies, the execution system employs monitoring procedures to make sure the execution of the plan does not divert from its intended course of action (Fikes, Hart, & Nilsson 1972), (DeGiacomo, Reiter, & Soutchanski 1998).

Execution monitoring approaches have generally focused on comparing the explicit effects of actions to what is really produced by the execution of the action (Haigh & Veloso 1997), (Pettersson 2005). This supposedly means that the effects to monitor are directly observable. That is of course not always realistic in a real world environment where checking expectations is a complex process. Therefore, more advanced forms of reasoning involving semantic knowledge were proposed to derive implicit expectations and monitor them (Bouguerra, Karlsson, & Saffiotti 2007b). For example, a mobile robot executing an action to enter an office should expect to see properties typical for an office, i.e., at least a desk, a chair, and possibly a PC. If the robot is entering a kitchen instead, it should expect to see an oven, a sink, etc. These implicit expectations are details that would add complexity to the initial planning task, if the planner has to reason explicitly about them. That is why they are encoded outside the actions of the task planner. In (Bouguerra, Karlsson, & Saffiotti 2007b) it is shown how the robot can compare its implicit expectations to the objects (and properties of objects) it actually observes, and come to one of three conclusions regarding the expectations: (1) they are confirmed (2) they are violated (3) it cannot be determined whether they hold or not, e.g., due to that only parts of the location or objects under observation can be seen at the moment. That brings us to the problem we tackle in this paper, i.e., expectations are not always immediately observable.

We extend the semantic knowledge-based execution monitoring to handle the third case above. The key idea is to model the problem of checking implicit expectations as a planning task, where the initial state represents the missing information situation and the goal represents a situation where that information is available. Therefore, the generated plan includes movement and observation actions needed to gather the required information. The implicit expectations to check are related to the execution of an action that is itself part of a top-level plan, which may have been generated by the same planner used for planning information gathering (but with a different domain description). In this work, however, it is planning information gathering for the purpose of monitoring that concerns us.

The reason why we opted for AI sensor-based planning is the ability to handle complex situations involving missing information in an effective and automatic way. Also, by combining semantic domain-knowledge and AI sensor-based planning, the proposed approach achieves an interleaving of planning and execution, which is a desired ability of autonomous robotic systems acting in uncertain environments(Nourbakhsh & Genesereth 1996). In fact, at task planning time, the task planner can reason on a more abstract level (office, kitchen, etc.) and rely on the monitoring process to check the details (desk, oven, etc.) at execution-time. The latter, in turn, uses sensor based planning when it is needed to do so.

In the next section we give an overview of the semantic knowledge-based execution monitoring framework. Then, we detail how checking expectations can be formulated as a planning task. Before concluding, we describe test scenarios run on a mobile robot.

*This work has been supported by the Swedish KK foundation.

Semantic Knowledge-Based Monitoring

Semantic knowledge refers to knowledge about objects, their classes and how they are related to each other (this knowledge is sometimes called "ontological" especially in the context of web contents). In the following we give a short overview of how such knowledge can be used in execution monitoring. For more information, the reader is referred to (Bouguerra, Karlsson, & Saffiotti 2007b).

The LOOM System

In this paper, we use LOOM (MacGregor 1999), a well established knowledge representation and reasoning system for representing and managing the semantic domain-information. The choice of LOOM was motivated by theoretical and practical considerations: LOOM is a well supported open source project whose knowledge representation and reasoning is based on description logics (DLs) (Baader *et al.* 2003) which provide a good trade-off between representation power and reasoning tractability. Another important characteristic of DLs is their reasoning capabilities of inferring implicit knowledge from the explicitly represented knowledge.

LOOM provides a language to write definitions of concepts and relations, and an assertion language to specify constraints on relations and concepts and to assert facts about individual objects.

Semantic knowledge in LOOM is organized in knowledge bases that contain definitions of concepts and relations between concepts. Concepts are used to specify the existence of classes of objects such as "there is a class of rooms" or "a bedroom is a room with at least one bed":

```
(defconcept Room)
(defconcept Bedroom
            :is (and Room (at-least 1 has-bed)
                      (at-most 1 has-sofa)))
```

The term `has-bed` in the second definition denotes a relation between objects of class `Bedroom` and objects of class `Bed`[1]. This relation is defined in LOOM as follows:

```
(defrelation has-bed
             :domain Bedroom :range  Bed)
```

When a relation `R` appears within a concept expression it is called a role. A filler of a role `R` is an object (individual) that corresponds to the second argument of `R` when considered as a binary predicate

The atomic constructs `(at-least 1 has-bed)` and `(at-most 1 has-sofa)` above specify constraints over the number of beds, respectively sofas, that can be in a bedroom. It is also possible to specify constraints over the types of objects an object can be in relation with. For instance, `(some R C)`, respectively `(all R C)`, is used to specify that at least one filler, respectively all fillers, of role `R` be of type `C`. The concept construct `(oneOf` $a_1, a_2, \cdots, a_m$`)` specifies a class of objects restricted to be in the set $\{a_1, a_2, \cdots, a_m\}$.

More complex concept expressions are constructed by combining other concept names using a limited number of connectives (`and, or, not, implies`). The semantics of concept expressions are interpreted in terms of set theory operations or in terms of equivalent first-order logic formulas over a non empty set of individuals.

Once the definitions are specified, specific individuals can be asserted to exist in the real world.For example:

```
(tell (Bedroom r1)(has-bed r1 b1))
```

asserts that `r1` is an instance of `Bedroom` and results in classifying `b1` as a bed (because the range of the relation `has-bed` is of type `Bed`). The instance `r1` is also classified (deduced) automatically as an instance of the class `Room`.

Classification is performed based on the definitions of concepts and relations to create a domain-specific taxonomy. The taxonomy is structured according to the superclass/subclass relationships that exist between entities. When new instances of objects are asserted (added to the knowledge base), they are classified into that taxonomy.

Monitoring Framework

The monitoring process proposed in (Bouguerra, Karlsson, & Saffiotti 2007b) uses semantic domain knowledge to derive implicit expectations related to the successful execution of a plan action.

Briefly, the monitoring process uses the action model to determine the assertions to monitor, which are positive effects involving objects that have some semantic properties. For instance the action `(enter r1)` has a positive effect `(robot-in = r1)`. If `r1` is asserted in the semantic knowledge base to be of type `Bedroom`, implicit expectations of being in a bedroom are derived for monitoring. On the other hand the action `(pick-up c1)` has positive effect `(holding c1)`. If `c1` is asserted to be of type `Cup`, then the implicit expectations to derive are properties of a cup, as well as properties specific to `c1` such as color, when available. However, it is not necessarily all implicit expectations that are of interest. In particular, we are only interested in observable properties and relations. For instance, the shape of an object may be observable, whereas what materials it is made of may be more difficult to observe. Therefore, we need to classify properties and relations into observable and non-observable.

The implicit expectations are then checked using the available perceptual information. For instance, if the robot has executed `(enter r1)`, where `r1` is asserted to be a bedroom, the monitoring module checks whether at least one bed has been perceived in the current room, to conclude that the action has been executed successfully.

The process of checking the implicit expectations gives rise to one of three outcomes:

1. All expectations hold. Therefore, *success* is returned to reflect that the expectations are verified.
2. At least one expectation does not hold. Thus the monitoring process returns *failure*. In this case, the robot might have to re-estimate the current world state and then replan from there to achieve the top-level task.
3. The truth values of some expectations are not known. This happens when there is missing information, due to occlusions for instance, needed to evaluate the expectations.

[1] As notation, the first letter of concept names is capitalized, while relation names start with uncapitalized letter.

The plan execution module proceeds to the execution of the next action of its plan only when all the expectations (implicit and explicit) are found to hold in the real world.

Planning for Execution Monitoring

In the third outcome above, the robot has two options. It can either be credulous and consider the absence of counter-evidence as sufficient grounds for assuming that the action succeeded. Or, it can take a cautious approach and actively try to find that missing information. Which approach to take may depend on many different factors, such as the prior probability of action failure. In the following, we assume a cautious approach, where the robot has to look for information required to evaluate expectations.

We propose an approach based on automatically analyzing and encoding the situation as a planning problem whose solution would involve generating an information gathering plan. The successful execution of the information gathering plan makes it possible to determine whether the expectations hold or not.

The generation and execution of monitoring plans requires information from different sources. First, we obviously need the semantic knowledge base in order to generate the implicit expectations. Second, a planning domain is needed, specifying among other things observations actions to test different properties. Other types of knowledge include spatial information such as topological maps and the spatial relations between objects (Cambon, Gravot, & Alami 2004). In the following we focus on the planning domain.

Modeling The Planning Domain

A planning domain consists of the specification of world states, the actions executable in each state, as well as the observations to be made in each state when executing a specific action. We use a first order language to encode world states and action templates.

Since the implicit expectations to evaluate are simply atomic concept constructs, the planning domain contains an observation action for each observable atomic concept construct. Each action gathers information so that the related concept construct can be evaluated. For instance:

- Action `(eval-<A> ?x)` collects information required to verify whether the individual bound to the variable `?x` is of type A. Thus, `(eval-room r10)` checks if the individual `r10` is an instance of the atomic concept `room`, and `(eval-container c1)` checks whether object `c1` is of type `container`. Note that there are as many action names as atomic concepts.
- Action `(eval-all R C ?x)` keeps track of the perceived individuals related (through R) to `?x`. It concludes that `(all R C)` is verified only if those individuals are *all* of type `C`.

Here is a detailed description, in the first order language of the PTLPLAN planner (Karlsson 2001), of the action `(eval-at-least ?n ?r ?x)` associated with the `at-least` atomic concept construct, i.e., `(at-least n R)` of an instance `i`. The variables `?n`, `?r`, and `?x` are to be bound respectively to `n`, `R`, and the instance `i`.

```
action:(eval-at-least ?n ?r ?x )
prec: (and (not (known (at-least ?n ?r ?x)))
           (robot-at = ?l)(part-of ?l = ?x)
           (not (checked ?r ?l))(can-check ?r ?l))
res:(and (checked ?r ?l = t)
         (cond ((at-least ?n ?r ?x)
                (obs (at-least ?n ?r ?x = t)) )
               ((and (at-least ?n ?r ?x = f)
                     (forall(?l)(can-check ?r ?l)(checked ?r ?l)))
                (obs (at-least ?n ?r ?x = f)))
               (t (and (obs (at-least  ?n ?r ?x = f))
                       (at-least ?n ?r ?x = t f))))
```

The intuition behind the action is that the robot can move between different positions, and at each position it can observe a number of individuals related to `?x` by `?r`. While doing that, it keeps track of the total number of observed individuals and compares it to `?n`.

The initially false predicate `(checked ?r ?l)` denotes whether the robot tried to observe individuals, needed to evaluate the relation `?r`, from position `?l`. The initial truth value of the predicate `(at-least ?n ?r ?x )` is unknown. The action is intended to observe the truth value of this predicate, to determine if the constraint `(at-least n R)` is true for the individual bound to `?x`. We use `(known` α`)` to denote that the formula α is true in all the possible worlds of the belief state where the action is applied.

In short, the precondition part specifies when the action is applicable, i.e., the truth value of `(at-least ?n ?r ?x)` is not known, and the robot is at a location where it is possible to observe individuals related to `?x` by relation `?r`. The results part encodes the effects of the action. Besides asserting `(checked ?r ?l = t)`, the action has also three conditional outcomes specified with the `cond` form. Note that the `cond` form works essentially like the Lisp `cond`, and the `obs` form is used to encode run-time observations.

1. The first outcome is observing that the constraint is verified. This happens when the predicate `(at-least ?n ?r ?x)` is true, i.e., at execution time, at least `?n` objects have so far been observed to be related to `?x` by `?r`.
2. The second outcome is observing that the constraint is violated. This is the result when the robot has visited all locations where it is possible to perceive objects that satisfy `?r`, yet their total number is still less than `?n`.
3. Finally, if the total number of perceived objects is less than `?n` and there are locations where the robot may observe extra objects, Then the third outcome is observing that `(at-least ?n ?r ?x)` is not verified and asserting its truth value to be unknown.

Planning Process

Generating plans to collect information successfully, requires that the planner takes into account the issue of partial observability of the environment. To this end, belief states are used to represent the agent's incomplete and uncertain knowledge about the world at some point in time, i.e., a belief state represents a set of hypotheses about the actual state of the world given past observations.

Initial Belief State Before calling the planner, the monitoring process formulates an initial belief state containing hypotheses about the truth value of each expectation to

check. This is done by asserting that the expectations can be true or false.

Example 1 Suppose that the robot has executed the action `(enter r1)`, where `r1` is an instance of `Bedroom` whose definition is given above. In the case where `r1` is known to be a room and the robot has yet not seen any sofa nor any bed inside `r1` (i.e, the expectations `(at-least 1 has-bed)` and `(at-most 1 has-sofa)` are not known to be true or false for the individual instance `r1`), this situation is encoded as:

```
(and (room r1)(at-least 1 has-bed r1 = t f)
     (at-most has-sofa 1 r1 = t f))
```

Where "`= t f`" means the truth value can be either `t`(rue) or `f`(alse). This formula encodes a belief state with four hypotheses (possible worlds).

The monitoring process adds to the initial belief state the locations where it is likely to observe individual objects or features that would bring more information about an unknown concept construct. For instance, the robot can decide that in order to look for beds, it needs to scan room `r1` from two locations `r1-1` and `r1-4`. Therefore it adds `(can-check bed r1-1)`,`(part-of r1-1 = r1)`,...etc to its belief state. In our current implementation this information is part of the map.

Goal Specification The goal formula of the planning problem is a modal knowledge formula containing a conjunct of the predicates associated with the expectations whose truth value are unknown. The predicates are assigned the expected truth value of their corresponding expectations. For the previous example, the goal formula is:

```
(known (and (at-least 1 has-bed r1 = t)
            (at-most 1 has-sofa r1 = t)))
```

Plan Generation In practice, we use the progressive planner PTLPLAN (Karlsson 2001). PTLPLAN searches in a space of belief states. Actions can both have causal effects that change properties in the world, and observation effects that reveal some of the hidden information about the exact state of the world. Hence, observations split up a belief state into several new and more informative belief states. The latter leads to conditional branches in the plan. The planner starts from the initial belief state and adds actions until a belief state satisfying the goal is reached. When an action results in several new belief states with different observations, the planner inserts a conditional branching in the plan and continues planning for each branch separately.

To be able to resume the execution of the top-level plan, information gathering plans are restricted to actions that do not alter the top-level plan state in any relevant way. For example, the information gathering plan to verify the expectations of being in `r1` is not allowed to include actions to move outside `r1`. In the scenarios we consider here, which only involve observation and movement actions and where the top-level actions are to move to a certain room, such a simple schema is sufficient. A more flexible approach is to require that certain conditions hold at the end of the plan execution, such as the robot being in the same room when the generation of the monitoring plan was launched.

Example 2 The following plan is generated for checking whether `r1` is a bedroom, starting from the situation where the truth values of the implicit expectations `(at-least 1 has-bed r1)` and `(at-most 1 has-sofa r1)` are unknown.

```
((eval-at-least 1 has-bed r1)
 (cond ((at-least 1 has-bed r1 = f)
        (move r1-2)(eval-at-most 1 has-sofa r1)
        (cond ((at-most 1 has-sofa r1 = t)
               (move r1-4)(eval-at-most 1 has-sofa r1)
               (cond ((at-most 1 has-sofa r1 = t)
                      (eval-at-least 1 has-bed r1 )
                      (cond ((at-least 1 has-bed r1 = f)(fail))
                             ((at-least 1 has-bed r1 = t)(success))))
                      ((at-most 1 has-sofa r1 = f)(fail))))
               ((at-most 1 has-sofa r1 = f) (fail))))
        ((at-least 1 has-bed r1 = t)
        (move r1-2)
        (eval-at-most 1 has-sofa r1 )
        (cond ((at-most 1 has-sofa r1 = t)
               (move r1-4) (eval-at-most 1 has-sofa r1)
               (cond ((at-most 1 has-sofa r1 = t) (success))
                      ((at-most 1 has-sofa r1 = f) (fail))))
               ((at-most 1 has-sofa r1= f) (fail) )))))
```

The special action `(fail)` (resp. `(success)`) denotes failure (resp. success) in satisfying the goal. Note that the plan declares failure as soon as the observation `(at-most 1 has-sofa r1)` evaluates to false, meaning that more than one sofa has been seen in `r1`.

The generated plan includes movement and observation actions. In fact that is specific to the navigation scenarios, and not a restriction. In other scenarios, movement actions might not be needed, but observation actions will always be necessary, since the aim is to gather information. For example, if the robot is executing the action `(grab c21)`, where the symbol `c21` refers to a cup that contains coffee, the observation plan would include actions to check the content of the cup but no robot movement actions.

Plan Execution

The execution of the information gathering plan is carried out by executing each action separately. Non observation actions are translated into low-level sensorimotoric processes that control the motion of the robot. Observation actions are translated into processes that use the newly available perceptual information to assert facts needed to evaluate the truth value of the predicate corresponding to the concept construct. For instance executing action `(eval-at-most 1 has-sofa r1)` results in adding information needed to evaluate the truth value of `(at-most 1 has-sofa r1)`. In this case, all newly perceived sofas, are added to the execution-time belief state. In other words the assertion `(and (sofa sf) (sofa r1 sf))` is executed for all newly perceived sofas `sf`.

The execution of the actions of the information gathering plan are also monitored by the same framework. This means that a new plan might be needed to check the execution of an observation action, resulting in recursive active monitoring (see test scenarios section). Note that there is no risk for infinite recursive information-gathering, as LOOM does not allow cyclic taxonomies.

The monitoring module concludes that the implicit expectations are verified, when the last executed action is `success`. Reaching the `fail` action implies that there was at least one violated expectation.

Test Scenarios

In order to test our approach, we implemented the monitoring framework on a Magellan Pro mobile robot (figure 2) running a fuzzy behavior control architecture, and using LOOM for knowledge representation and reasoning, and PTLPLAN to generate both task plans and information gathering plans (in all the test scenarios we ran, the planning time was less than one second.)

Figure 1 shows the main components of the architecture implemented on our mobile robot. Briefly, these components are:

- The planning component includes the planning engines and the different planning domain definitions. It responds to planning requests sent by the executor. It also communicates the model of the action under execution to the monitor component.
- The executor component keeps track of the current plan in execution and carries out the process of translating the high-level symbolic plan actions into executable processes. It also responds to user's requests, e.g., to achieve a new task, or to interrupt the execution of the current plan under execution.
- The monitor component performs monitoring at the different levels of execution. This includes the high-level semantic knowledge-based monitoring. The monitor process receives the action to monitor from the executor and reports back the status of the action execution. In case the monitoring process needs some information to be collected, the planning component is requested to generate a plan to achieve the task of information gathering.
- The perception component establishes and maintains the correspondence between the percepts produced by the onboard sensing modalities (mainly vision and olfaction) and the symbols used by the planning and semantic knowledge systems.
- The state component keeps track of the execution context of plans as well as maintains the expected state of the world and the robot (self-localization). It is updated by the executor and the perception components to reflect action explicit effects and what the robot is observing.
- The semantic knowledge base component (SKB) consists of the knowledge representation and management system LOOM. Its role is storing conceptual and assertional domain knowledge. It also processes and answers queries, e.g., coming from the monitoring module.

As our robot cannot perform manipulation tasks, our test scenarios consisted in performing mostly navigation tasks. Since our main objective is to show the capacity of monitoring using semantic knowledge and not object recognition, we let simple shapes like balls and boxes stand in for beds, sofas, etc. The experiments have been performed in a lab imitated-house, placing the simple objects to simulate pieces of furniture in the different rooms (see figure 2).

The semantic knowledge base contains among other things the following concept definitions:

```
(defconcept Room)(defconcept Bed)(defconcept Oven)
(defconcept Sink)(defconcept Sofa)(defconcept TV)
(defconcept Bedroom :is
            (:and Room (at-least 1 has-bed)
            (at-most  1 has-sofa)))
(defconcept Kitchen :is
           (:and Room (exactly 1 has-sink)
                (at-least 1 has-oven)
                (exactly 0 has-bed)(exactly 0 has-sofa)))
(defconcept Living-room :is
           (:and Room (at-least 1 has-sofa)
                 (exactly 1 has-tv)(exactly 0 has-sink)))

(tell (Bedroom r1)(Bedroom r2)(Living-room r3)(Kitchen r4))
```

Passive Monitoring. In this test scenario, the robot moved into room `r4` asserted to be the kitchen. In order to verify that the robot was in the correct room (not dislocated), the monitoring module derived expectations of being in a kitchen. As `r4` is a small room, this could be determined without any further movement, i.e., without calling the planner. The monitoring module was able to immediately establish the truth value (true or false) of the implicit expectations.

Active Monitoring. The aim of this test scenario was to show how planning could be used to collect information to infer the truth value of implicit expectations. The assigned top-level task was to clean the living-room (`r3` in figure 2), starting from the kitchen `r4`. The top-level task plan `(enter r3);(clean r3)` was produced by the task planner to accomplish the assigned task. Upon finishing the execution of the first action, i.e., after having moved to destination `r3`, the monitoring process was triggered to check both the explicit and implicit expectations. The implicit expectations were derived from the DL definition of the concept `living-room`, i.e., `(and room (at-least 1 has-sofa )(exactly 1 has-tv)(exactly 0 has-sink))`. Evaluation of these implicit expectations using the immediately available perception information revealed that only the expectation `(room r3)` was verified. The truth value of the other three expectations was not known. Consequently, the monitoring process created a belief state reflecting the situation and called the planner to find a plan to check whether `r3` has at least one sofa, at least one TV set, and no sink. The generated plan included actions to move and scan the room from four predetermined locations looking for sofas, TV-sets, and sinks. The planning problem was solved in less than 1 second. We had two runs of this experiment. In the first run, the room was cor-

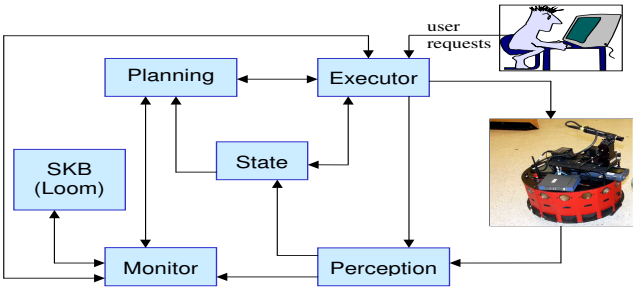

Figure 1: Architecture components

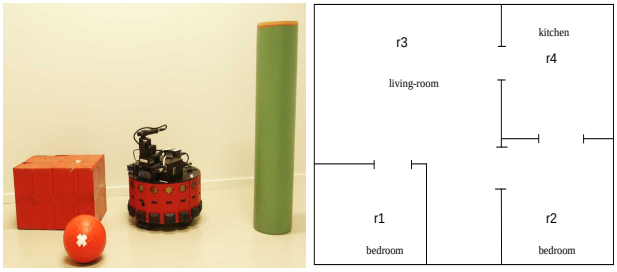

Figure 2: Experimental setup. (Left) Our robot with simple objects used to represent furniture items. (Right) map of the environment used in our experiments.

rectly found to be a living-room, thus the action `(enter r3)` was concluded to have been executed successfully and the execution of the top-level task plan was resumed with the next action, i.e., `(clean r3)`. In the second run, we modified the room by adding an object of type sink and removing the objects representing the sofa and the TV-set. As a result, the execution of the information gathering plan failed to find evidence that the room was a living-room. Consequently, the monitoring process concluded that the action had failed to execute successfully.

Recursive Monitoring. In this test scenario, we show how the framework applies recursively to monitor the execution of an information gathering plan (itself generated to monitor expectations of a top-level plan). We also show how our approach applies to actions other than navigation. We modified the semantic knowledge base by redefining the range of `sofa` relation to be the `two-seat-sofa` concept given as:

```
(defconcept Two-Seat-Sofa :is
            (and Sofa (= number-of-seats 2)))
```

The robot executed a task plan that included an action to enter the bedroom `(enter r1)` which resulted in the generation and execution of the information gathering plan above. Using semantic knowledge to monitor the execution of `(eval-at-most 1 has-sofa r1)` action involved deriving the implicit expectation that any perceived sofa inside `r1` had to verify the constraint `(= number-of-seats 2)`. Consequently, a new information gathering plan was generated, every time a new sofa was perceived, to check whether it had a number of seats equal to 2. Each plan consisted of moving in front of the sofa and observing the number of its seats.

This test scenario shows that our approach leads to a form of interleaving of planning and execution. In fact, in the first plan above, the planner did not include any actions to check the number of seats of a sofa. This was handled at run-time by the monitoring process once a sofa was perceived, i.e., by generating an information gathering plan to check that the sofa is a two-seater

Simulated Information Gathering: In this test scenario, we show that planning for monitoring can be used to check the expectations of actions that do not involve robot location. The top-level task plan included the action `(pick-up c1)` to pick-up cup `c1`, where a cup is defined in the semantic knowledge base as:

```
(defconcept Cup :is
            (and Container (exactly 1 has-handle)))
```

Once the robot had finished the execution of the action, the monitoring process was called to check that it was executed successfully. This meant that the robot should be holding an object (explicit effect) and that the held object should be a container with one handle (implicit expectations). Checking the explicit expectations gave a positive answer. However using the available perceptual information about the object the robot was holding, it was not possible to establish whether that object was a cup. This triggered information gathering planning to find out whether the held object was a container that had exactly one handle. The generated plan was as follows(due to practical reasons, we only simulated the execution of the resulting plan):

```
((move-gripper p1)(eval-container c1)
 (cond ((container c1 = t)(eval-at-most 1 has-handle c1)
         (cond ((at-most 1 has-handle c1 = t) (success))
               ((at-most 1 has-handle c1 = f) (fail))))
        ((container c1 = f) (fail))))
```

The intended effect of `(move-gripper p1)` was moving the robot's gripper to a position `p1` in front of the camera, so the observation actions `(eval-container c1)` and `(eval-at-most 1 has-handle c1)` could be executed (the observation actions templates were different from those involving robot movement).

Related Work

Although there is a considerable amount of work related to plan execution monitoring in mobile robotics (Pettersson 2005), to the best of our knowledge, no research work has used semantic knowledge and active information gathering to monitor plan execution.

Reactive planning architectures, such as PRS (Ingrand, Georgeff, & Rao 1992) use hand-coded procedures for monitoring the events that might affect the execution of their plan actions. Consequently, expectations are explicitly coded in the monitoring procedure, which makes monitoring not flexible. In plan-based mobile-robotic architectures, such as Shakey (Fikes, Hart, & Nilsson 1972) and the LAAS architecture (Alami *et al.* 1998), monitoring amounts to looking for discrepancies between the predicted state based on the explicit effects of actions, and the real world state as computed by the on-board sensing modalities.

There are several works that address information gathering using planning. In mobile robotics, sensor-based planning was used to collect information to recover from perceptual failures (Bouguerra, Karlsson, & Saffiotti 2006). Information collecting actions are also part of sub-symbolic policies for robot navigation (Simmons & Koenig 1995), and localization (López *et al.* 2005).

It is worth emphasizing, that we are not using description logics to model planning domains and problems as in (DeGiacomo *et al.* 1997) and (Badea 1998). Instead, we are using planning to collect information to monitor information derived from semantic knowledge.

Conclusions

We have presented an intelligent plan execution monitoring approach combining semantic domain knowledge and sensor-based planning. We believe that execution monitoring is a complex task, and therefore it needs to be addressed with powerful reasoning tools. Semantic knowledge was shown to be useful for deriving implicit expectations of action execution. On the other hand, sensor-based planning was employed to actively collect information required to determine the truth value of such expectations. As a result, the proposed monitoring process is flexible and effective, since unpredictable and complex situations are handled during run-time. We also argue that the proposed approach is adequate for environments where lack of information is an inherent feature, since at planning time only abstract representation of the planning domain can be used. More details are handled at run-time by the monitoring process.

We would like to emphasize that what is presented here is a first version of active semantic execution monitoring and there is potential to develop the method further, e.g., by using other knowledge representation formalisms (including probabilistic ones).

Finally, there are open issues that we plan to address in our future work. First, a deeper experimental validation that addresses scalability and richer domains is among our objectives. Second, the issue of when to engage in active information gathering needs to be investigated carefully, since there might be situations where information gathering is expensive. We also regard the issue of which expectations should be selected for checking as an important one, as the number of expectations might be very big. In a related work (Bouguerra, Karlsson, & Saffiotti 2007a), we have considered a probabilistic approach to semantic knowledge-based execution monitoring, which computes probabilities for different action outcomes, e.g., different locations of the robot. It is based on both the a priori probability of each outcome (e.g., the probability of navigating successfully to the desired room), and the probability that the current place or object is actually consistent with the semantic knowledge given what is observed. This approach can support a more informed decision about whether more information is needed or not and what observable properties of objects are more important to check than others. In addition, it would permit us to take advantage of the ability of PTLPLAN to deal with probabilities, and to compute an expected cost of the monitoring plan.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *Int. Journal of Robotics Research* 17(4):315–337.

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook*. Cambridge University Press.

Badea, L. 1998. Planning in description logics: Deduction versus satisfiability testing. In *Proc. of the 13th European Conf. on AI*, 479–483.

Bouguerra, A.; Karlsson, L.; and Saffiotti, A. 2006. Situation assessment for sensor-based recovery planning. In *Proc. of the 17th European Conf. on AI*, 673–677.

Bouguerra, A.; Karlsson, L.; and Saffiotti, A. 2007a. Handling uncertainty in semantic-knowledge based execution monitoring. In *In Proc. of 2007 IEEE Int. Conf. on Intelligent Robots and Systems*.

Bouguerra, A.; Karlsson, L.; and Saffiotti, A. 2007b. Semantic knowledge-based execution monitoring for mobile robots. In *In Proc. of 2007 IEEE Int. Conf. on Robotics and Automation*, 3693–3698.

Cambon, S.; Gravot, F.; and Alami, R. 2004. A robot task planner that merges symbolic and geometric reasoning. In *Proc. of the 16th Eureopean Conference on Artificial Intelligence*, 895–899.

DeGiacomo, G.; Iocchi, L.; Nardi, D.; and Rosati, R. 1997. Planning with sensing for a mobile robot. In *Proc. of 4th European Conf. on Planning*, 156–168.

DeGiacomo, G.; Reiter, R.; and Soutchanski, M. 1998. Execution monitoring of high-level robot programs. In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning*, 453–465.

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

Haigh, K. Z., and Veloso, M. M. 1997. High-level planning and low-level execution: Towards a complete robotic agent. In *Proc. of the 1st Int. Conf. on Autonomous Agents*, 363–370.

Ingrand, F. F.; Georgeff, M. P.; and Rao, A. S. 1992. An architecture for real-time reasoning and system control. *IEEE Expert* 7(6):34–44.

Karlsson, L. 2001. Conditional progressive planning under uncertainty. In *Proc. of the 17th Int. Joint Conf. on AI*, 431–438.

López, M. E.; Bergasa, L. M.; Barea, R.; and Escudero, M. S. 2005. A navigation system for assistant robots using visually augmented pomdps. *Autonomous Robots* 19(1):67–87.

MacGregor, R. 1999. Retrospective on loom. Technical report, Information Sciences Institute, University of Southern California.

Nourbakhsh, I., and Genesereth, M. 1996. Assumptive planning and execution: a simple, working robot architecture. *Autonomous Robots Journal* 3(1):49–67.

Pettersson, O. 2005. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems* 53(2):73–88.

Simmons, R., and Koenig, S. 1995. Probabilistic robot navigation in partially observable environments. In *Proc. of the Int. Joint Conf. on AI*, 1080–1087.