Failure Prediction based on Monitoring Sequences of Actions and Action Duration

Giovani Farias[1], Ramon Fraga Pereira[1], Lucas Hilgert[1], Felipe Meneguzzi[1], Renata Vieira[1], and Rafael H. Bordini[1]

[1]Pontifical Catholic University of Rio Grande do Sul – PUCRS
School of Informatics – FACIN – Porto Alegre, Brazil

{giovani.farias,ramon.pereira}@acad.pucrs.br
lucaswhilgert@gmail.com
{felipe.meneguzzi,renata.vieira,rafael.bordini}@pucrs.br

Abstract. *An agent can attempt to achieve multiple goals and each goal can be achieved by applying various different plans. Anticipating failures in agent plan execution is important to enable an agent to develop strategies to avoid or circumvent such failures, allowing the agent to achieve its goal. Plan recognition can be used to infer which plans are being executed from observations of sequences of activities being performed by an agent. Symbolic Plan Recognition is an algorithm that represents knowledge about the agents under observation in the form of a plan library. In this paper, we use this symbolic algorithm to find out which plan the agent is performing and we develop a failure prediction system, based on information available in the plan library and in a simplified calendar which manages the goals the agent has to achieve. This failure predictor is able to monitor the sequence of agent actions and detects if an action is taking too long or does not match the plan that the agent was expected to be performing.*

1. Introduction

Recently, the number of real-world applications that deal with the need to recognise goals and plans from agent observations is on the rise. These applications can be found in fields such as human assisted living [Masato 2012], interface agent systems [Armentano and Amandi 2007], human-computer interaction [Hong 2001], traffic monitoring [Pynadath and Wellman 1995], and others. However, techniques that include the task of anticipating failures during agent plan execution have received relatively little attention. Multi-agent environments are dynamic since they are in a constant estate of change resulting from agents' actions. When these changes occur, a plan that was expected to work before, may fail. Thus, anticipating from agent observations when a plan is going to fail can be an important mechanism during the plan recognition process. Plan recognition approaches often do not make such inferences, which means that when an agent has no intention to complete or finish a plan these approaches continuously attempt to recognise what the agent is doing. In daily activities most people interrupt the plan that they are performing for some reason, such as, getting their attention drawn to something else, getting distracted by other events, or being interrupted by an emergency that needs immediate attention. In a plan recognition context, we consider that a plan is going to fail when the sequence of actions is taking too long or does not match the plan which the

observed agent should be performing at the moment. Our approach uses a calendar for managing some of the agent's goals over the near future, and when that information is available it facilitates our failure checking procedure, as well as plan recognition disambiguation. A plan failure can occur when an agent interrupts its current plan execution due to concurrent plans that need attention, or when an agent has to deal with conflicting plans. In this case, a mechanism to anticipate failures during agent plan execution can be useful in several situations, for example, helping an agent stay focused on a particular plan, or detecting when an agent is deviating from its regular activities.

Research on planning has focused on the modelling of actions with duration and stochastic outcomes, both theoretically as variants of Markov Decision Processes (MDP) [Mausam and Weld 2008], and domain description languages that express temporal planning (e.g., PDDL 2.1 [Fox and Long 2003], an extension of PDDL. In the literature, a similar approach to failure prediction, as we introduce in this paper, is plan abandonment detection. Geib and Goldman [Geib and Goldman 2003] proposes a formal model to recognise goal/plan abandonment in the plan recognition context, based on the PHATT (Probabilistic Hostile Agent Task Tracker) model [Goldman et al. 1999]. This formal model estimates the probability that a set of observed actions in sequence contributes to the goal being monitored, furthermore, Geib [Geib 2002] addresses some issues and requirements for dealing with plan abandonment, as well as intention recognition in the elderly-care domain.

In this paper we develop an approach to predict plan failure by monitoring agent's actions during its plan execution. Essentially, our approach to plan failure prediction features a mechanism that is composed of three modules. The first module is responsible for recognising the plan that the observed agent is executing. The second module checks if plans assigned to observed agent are being executed as scheduled in a predefined calendar. Lastly, the third module checks if actions are being executed as expected (i.e., not taking too long, and matching the current monitored plan). Thus, this approach can be used in complex software system, including health-care applications to improve functionality, such as activity recognition and task reallocation [Panisson et al. 2015] among agents representing human users, who collaborate to take care of a patient, by detecting if a person responsible for some activity of the patient is following his scheduled appointments; detecting problems, that may prevent the person in charge to attend to his obligations and send warning to the system.

2. Plan Recognition

Plan recognition can be defined as the task of recognising the intentions of an agent based on the available evidence, that is, agent actions, explicit statements about intentions, and agent preferences [Kautz and Allen 1986]. Plan recognition focuses on mechanisms for recognising the unobservable state of an agent, given observations of its interaction with its environment. In other words, a plan recognition system must have a mechanism that is capable of inferring agent intentions by observing the agent actions in the environment. This mechanism retrieves, from a given set of observations, one or more hypotheses of the agent's current plan of action. The practical knowledge used to infer plans is domain dependent and, therefore, is commonly specified beforehand for each specific domain. This domain dependent information is usually encoded as two parts of inputs for the recogniser: a set of observed actions and a set of plans and goals. More specifically, the inputs

to a plan recogniser are a set of goals that the recogniser expects the agent to carry out in the domain, a set of plans describing the way in which the agent can reach each goal, and a sequence of actions currently being performed by the observed agent (i.e, observations of agent actions). Thus, the plan recognition process itself consists in inferring the agent plan and determining how the observed actions contribute to performing it.

Symbolic plan recognition is a type of plan recognition mechanism that narrows the set of candidate intentions by eliminating the plans that are incompatible with current agent actions. Plans make up a plan library and can include preconditions, effects, and sub-goals. Generally, symbolic approaches assume that the observer has complete knowledge of the agent's possible plans and goals. Symbolic approaches handle the problem of plan recognition by determining which set of goals is consistent with the observed actions. Algorithms to recognise the intentions and plans executed by autonomous agents have long been studied in the Artificial Intelligence field under the general term of *plan recognition*. Kautz and Allen [Kautz and Allen 1986] focus on symbolic methods providing a formal theory of plan recognition. Usually, these approaches specify a plan library as an action hierarchy in which plans are represented as a plan graph with top-level actions as root nodes, and plan recognition is then reduced to a graph covering problem. The plan recognition process attempts to find a minimal set of top plans that explain the observations. For a good overview of plan recognition in general, see Carberry [Carberry 2001], and for the most recent research in the field of plan, intention, and activity recognition, see Sukthankar et al. [Sukthankar et al. 2014].

3. Symbolic Plan Recognition

The *Symbolic Plan Recognition* (SBR) [Avrahami-Zilberbrand and Kaminka 2005] is a method for complete, symbolic plan recognition that uses a plan library, which encodes agent knowledge in the form of plans. SBR extracts coherent hypotheses from a multi-featured observation sequence using a *Feature Decision Tree* (FDT) to efficiently match these observations to plan steps[1] in a plan library. An FDT is a decision tree, where each node represents an observable feature and each branch represents one possible value of this feature. Determining all matching plans from a set of observations features is efficiently achieved by traversing the FDT top-down until a leaf node is reached. Each leaf node is a pointer to a plan step in the plan library.

A plan library is represented by a single-root directed acyclic connected graph, which includes all possible plans that an observed agent may execute. The term plan is used here in a broader sense, representing behaviours, reaction plans, and recipes. Typically, a plan library has a single root node in which its children are top-level plans and all other nodes are simply plan steps. Furthermore, in a plan library, *sequential edges* specify the expected temporal order of a plan execution sequence and *decomposition edges* decompose plan steps into alternative sub-steps. The plan library has no hierarchical cycles. However, plans may have a (sequential) self-cycle, allowing a plan step to be executed during multiple subsequent time stamps. Each agent action generates a set of conditions on observable features that are associated with that action. When these conditions are included, the observations match particular plan steps. Figure 1 shows a plan library example based on Activities of Daily Living (ADL), which is a term used

[1] In this paper, we use "plan step" as a synonym for "action".

in health-care to refer to daily self-care activities of people. This plan library shows sequential links represented by dashed arrows and decomposition links represented by solid arrows. For instance, there is a decomposition link between *managing-medication* and *getting-up*, and a sequential link between *getting-up* and *using-bathroom*. The top-level plans are *managing-medication*, and *leisure*. Figure 1 does not show the set of conditions on observable features associated with plan steps, and circled numbers denote time stamps (e.g., *using-bathroom* has been considered a hypothesis at time stamp 2). The decomposition edges are shown only to the first child plan. Thus, the path *root* $\rightarrow$ *managing-medication* $\rightarrow$ *having-lunch* $\rightarrow$ *at-kitchen* $\rightarrow$ *taking-medication* can be a hypothesis for the current plan being executed by an observed agent.

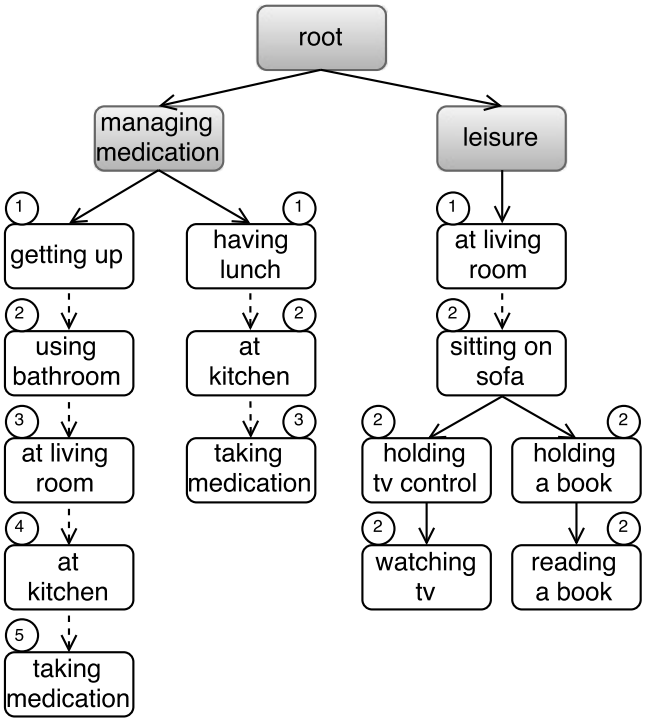

Figure 1. Example of plan library based on ADL.

4. Failure Predictor Components

The failure predictor is responsible for predicting plan failures during execution of an agent goal, more specifically, it tracks the execution of a goal and attempts to identify elements which can lead it to fail, for example, an action taking significantly more time than expected to conclude. The failure predictor is composed of three modules: the `SBR` module, which is responsible for recognising the plan that is being executed by the observed agent; an `Appointment Controller` module, which checks if the goals that are known to have been assigned to the agent are being executed as scheduled; and a `Plan-Step Controller` module, that checks if the plan steps (that compose the plan) are being executed as expected. These modules are presented, respectively, in Subsections 4.1, 4.2, and 4.3.

4.1. SBR Component

The `SBR` component implements the symbolic recogniser presented in Section 3. It is responsible for recognising the plan that an agent is currently executing, and it generates hypotheses about possible plans while the recognition is still not possible. This information is represented both as a list of candidate plans and as a hypotheses graph. As input, the `SBR` component receives *observations*, i.e., sets of contextual information about the observed agent and its actions. Examples of observations include the agent's global positioning coordinates, whether or not the agent is moving, or whether the agent is approaching a particular place, and any other contextual information that can be generated by an activity recognition process. As output, this component provides both the list of candidate plans and the hypotheses graph.

4.2. Appointment Controller

A plan library contains all known plans (agent goals) for a given domain, together with the sequence of actions that compose them, however, it does not define the time that an agent is expected to execute each plan, neither does it contain the time interval in which the plans have to be executed. These are essential information for the system to ensure that plans are being executed in an appropriate manner and to be able to detect potential failures in plan execution. The `Appointment Controller` component implements a simplified calendar which manages the agent goals and plans, it defines which plans of the plan library an agent is known to be responsible for, and at which time the agent is expected to execute some of them (to the extent that this is known in particular domains). This component also helps in disambiguation of candidate plans and in the early prediction of plan failures. It should be noted that only domain-related plans are kept in this individual calendar. Each entry (i.e., agent goal) in the `Appointment Controller` is composed of the following information:

- **Starting date** – Date in which the goal or plan is expected to be started;
- **Ending date** – Date in which the goal or plan is expected to end;
- **Title** – Title of goal or plan (e.g., *"managing-medication"*);
- **Description** (Optional) – Brief textual description of the goal (e.g., *"take medicine on time"*);
- **Related Plan ID** – Unique identifier of the relevant plan (i.e., the plan to achieve this goal), which corresponds to a top plan in the plan library (e.g., *"id:p1"*);
- **Tolerance** – Margin of error for the beginning and end times of each goal, e.g., some goals can start or end 5 minutes before (or after) the time for which it was originally scheduled without danger of the plan failing. This tolerance is necessary because, in real-world situations, goals usually do not start (or end) at the exact scheduled time.

Regarding schedule times, both the starting and ending dates of the goals are composed of day, month, year, hour, and minute (smaller units such as second, for example, are not necessary). The *tolerance* interval can be expressed in various time measures (e.g., hours or minutes).

4.3. Plan-Step Controller

The `Plan-Step Controller` monitors and analyses the plan execution (sequence of actions) to detect anomalies that can lead the plan to fail. The information necessary

for the `Plan-Step Controller` to operate is obtained through the `SBR` component, which provides information about the current plan and actions being performed; the plan library; and the file that contains information about expected time for each action execution. The plan library contains the known plans and the actions which need to be performed in a given plan for it finish successfully, besides the sequence in which these actions must be performed for a plan to be considered completed. However, it does not define when a plan should finish, neither the time in which actions must be executed. This type of information is important to detect anomalous behaviour during plan execution, such as:

- **Plan Interruption** – The plan execution is interrupted without all actions being completed;
- **Time Exceeded** – When an action takes significantly more time than expected, this typically leads to plan failure (e.g., being in a traffic jam);
- **Inconsistent Sequence** – The sequence of observed actions is inconsistent with the expected plan path in the plan library.

It is important to keep track of the actions being performed in order to be able to predict whether a plan is following the expected execution path. Thus, it is possible to identify a probable failure in plan execution and generate the required warnings according to failure type. The entry in file with data about expected execution time of each action is composed of the following information:

- **Plan-Step ID** – Unique identifier of related action, which corresponds to a plan step in the plan library (e.g., *"p1.11"*);
- **Label** – A label for identification the action (plan step) from the plan library (e.g., *"taking-medication"*);
- **Time** – Time that an action often take to be performed;
- **Tolerance** – A value whereby the ending time of the action is allowed to be delayed. For instance, an action can take 5 minutes in addition to its normal time to be performed. This tolerance is important for a real-world situation where actions can often take more time to be performed than an exact specified time.

5. Component Integration

The failure predictor components are integrated as presented in Figure 2. When the `SBR` (Section 3) is not able to determine the current plan (no plan or multiple plans were recognised) the `Appointment Controller` component is consulted (using the output of the plan recogniser). First, the component checks if there are plans scheduled for the moment in which it was consulted and, later, if a scheduled plan is in the list of candidate plans. During this verification the following situations might occur:

- There is no agent goal scheduled for the current time. In this situation, the `Appointment Controller` component has nothing to do, so the main cycle ends and the system awaits for a new observation;
- There is a plan scheduled for the current time, however, the candidate plan list is empty (no plans were recognised). In this case, the `Appointment Controller` component detects a failure in the scheduled goal execution (i.e., the goal that was expected to be executed at time the calendar was consulted) and a warning must be sent to the system, which should be able to handle this plan failure;

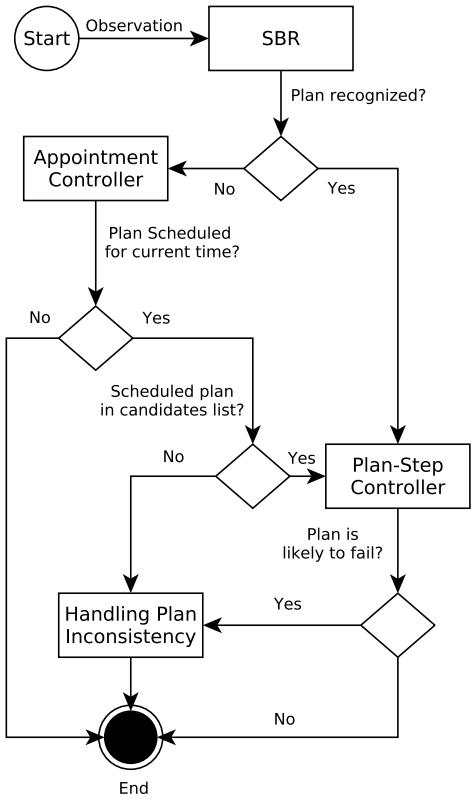

Figure 2. Components Integration.

- There is a plan scheduled for the current time and there are multiple plans in candidate plans list. In this case, the `Appointment Controller` verifies if the plan related to scheduled goal is present in plan candidate list. During this verification two situations might happen:
  - The plan related to the scheduled goal is in candidate plans list, thus, the referred plan is assumed to be the one that is currently being executed by agent;
  - The plan related to the scheduled goal is not in candidate plans list. In this situation, a failure is detected in the scheduled goal execution as it is not being executed by the agent as it should. Then, a warning related to the scheduled goal is sent to the `Handling Plan Inconsistency` step of the main cycle, in which the system will handle the plan related to the failing goal.

Regarding, goal scheduling in this initial implementation does not allow overlapping of goals (i.e., goals with coincident time intervals). That is, a new goal will not be added to calendar if it overlaps with an existing one. However, there is an exception for the overlapping rule regarding the starting and ending times. Two goals (A and B) are not considered as overlapping if the starting time of A is equal to the ending time of B. This exception is convenient, as sequential plans are usually scheduled with no time interval among them, e.g., A (1:00 p.m. to 2:00 p.m.) and B (2:00 p.m. to 4:00 p.m.) or A (4:00 p.m. to 5:00 p.m.) and B (2:00 p.m. to 4:00 p.m.).

The `Plan-Step Controller` is unable to disambiguate the list of candidate plans, thus, both `SBR` and `Appointment Controller` must inform only one goal and one plan step in each iteration with it. The `planController` (Algorithm 1) is part

of this component and responsible for handling such information, using the plan library and the data file with the plan step running times, in order to analyse the plan sequence (based on plan steps) and monitor the running time of each action. In this manner, it is possible to detect and report possible changes in plan execution in order to avoid possible failures.

Algorithm 1
`planController`(Current Goal g, Current PS p, List l)

```
1: Get plan step duration from l;
2: analyseCurrentGoal(g);
3: analyseCurrentPlanStep(p);
4: Inform possible failure;
```

Monitoring of the current plan is performed by the `analyseCurrentGoal` (Algorithm 2). Initially, the current goal is updated based on information received by `SBR` (Line 1) and a test is performed to check if it is a valid value (Line 2), after that, the algorithm checks if the current goal is equal to the previous goal (Line 5) that the agent was trying to achieve. If they are the same, it means agent is still carrying out the actions to achieve it, otherwise, the agent started to perform a new goal with a new plan. In this case, the algorithm has to verify if the previous goal was achieved successfully (Line 11) and the information in the plan library is used to check if the last plan step (of the previous goal) is a leaf node. If this is the case, it means that the plan was fully executed and probably has finished successfully, otherwise, the agent may have stopped performing the plan before its end or the agent is executing more than one plan at the same time, thus, the algorithm should send a warning about this possible failure (Line 15).

Algorithm 2
`analyseCurrentGoal`(Current Goal g)

```
1: current_goal ← g;
2: if current_goal = null then
3:    No goal can be checked;
4: else
5:    if current_goal = previous_goal then
6:       Goal g keeps running;
7:    else
8:       if previous_goal = null then
9:          Goal g started;
10:      else
11:         if previous_goal has finished in a leaf node then
12:            previous_goal finished and g started;
13:         else
14:            Goal g started;
15:            previous_goal stopped before ending;
```

The `analyseCurrentPlanStep` (Algorithm 3) is responsible for monitoring the execution of each action related to a goal, i.e., it analyses the sequence of execution

and the run time of each plan step. Initially, the current plan step is updated based on information received by `SBR`. The consistency of this information is checked and the algorithm then checks if the current plan step is equal to the previous plan step which the agent was performing (Line 5). If they are equal, it means that the agent is still performing the same plan step, thus, the algorithm has to check if the current action is within the time specified in the data file with the plan step running times. The `checkExecutionTime` (Line 7) receives as input the current plan step and checks if its run time is within the specified time, taking into account the specified tolerance for each action. If the agent is taking too long to perform some action, the algorithm detects this as unexpected behaviour and warns the system.

Algorithm 3
`analyseCurrentPlanStep`(Current Plan Step p)

```
1:  current_plan_step ← p;
2:  if current_plan_step = null then
3:     No plan step can be checked;
4:  else
5:     if current_plan_step = previous_plan_step then
6:        current_plan_step keeps running;
7:        checkExecutionTime(current_plan_step);
8:     else
9:        if isValidSequence(previous_plan_step, current_plan_step) then
10:          Current execution path is right;
11:       else
12:          Current execution path has changed;
13:       Update current_goal start time;
14:       previous_plan_step ← current_plan_step;
```

When the failure predictor detects a new action being performed by an agent, it is necessary to check if this action is part of the sequence of actions needed to accomplish the current goal (Line 9). The `isValidSequence` (Algorithm 4) uses the information in the plan library to check if the current plan step is part of a valid sequence of actions to achieve the current goal, receiving as input the previous plan step and the current plan step. If the current plan step contains a sequential parent node and this parent node is the previous plan step, it means the execution path is correct, otherwise, the current plan step does not match the execution path done so far to achieve the current goal.

Detecting whether the sequence of execution is valid is more complicated when the current plan step has a decomposition parent node, because the previous plan step does not have to be a parent node of the current plan step, but only be part of the current plan execution and follow the temporal order of the execution path. The `isPreviousNode` (Algorithm 5) algorithm receives as input the previous plan step and the current plan step. It checks the entire running sequence from current plan step node to previous plan step node in order to analyse the temporal order to determine if the current plan step is a valid sequence for current goal execution.

Algorithm 4
`isValidSequence`(Parent Node $parent$, Child Node $child$)

```
1: if child has a sequential parent then
2:    seq_parent ← child sequential parent;
3:    if seq_parent ≠ parent then
4:       return false;
5:    else
6:       return true;
7: else if child has a decomposition parent then
8:    dec_parent ← child decomposition parent;
9:    if dec_parent = parent then
10:      return true;
11:   else
12:      if isPreviousNode(parent, child) then
13:         return true;
14:      else
15:         return false;
16: else
17:   return false;
```

Algorithm 5
`isPreviousNode`(Parent Node $parent$, Child Node $child$)

```
1: if child has a sequential parent then
2:    seq_parent ← child sequential parent;
3:    if seq_parent = parent then
4:       return true;
5:    else
6:       return isPreviousNode(parent, seq_parent);
7: else if child has a decomposition parent then
8:    dec_parent ← child decomposition parent;
9:    if dec_parent = parent then
10:      return true;
11:   else
12:      return isPreviousNode(parent, dec_parent);
13: else
14:   return false;
```

6. Experiments

The objective of experiments is to show how our approach provides helpful reminders by monitoring and anticipating plan failure from agent observations, in this way, we model part of a scenario that represents agent behaviour based on Activities of Daily Living. These activities correspond to user single actions (e.g., *getting-up*, *watching-tv*, *reading-a-book*, *taking-medication*, *using-bathroom*), in this scenario, there is a person with disabilities who lives alone and needs constant monitoring to perform his daily activities, using plan library formalism, we model a set of plans for representing possible behaviour of this person, where some of these plans are shown in Figure 1.

To exemplify how our approach works, we schedule the top-level plan *managing-medication* to be performed between 7:00 a.m. and 7:30 a.m., in which, this schedule information is in the calendar (`Appointment Controller`). Considering the current part of the day as early morning, the sequence of activities to be performed in order to accomplish the plan *managing-medication* is: *getting-up* → *using-bathroom* → *at-living-room* → *at-kitchen* → *taking-medication*. According to this sequence, the user must move through the living room (i.e., plan step *at-living-room*) to complete the plan, however, this plan step is also part of the top-level plan *leisure*, in this case, the `SBR` component returns both top-level plans *managing-medication* and *leisure* when the current plan step is *at-living-room*. To deal with this ambiguity, our approach uses the `Appointment Controller` component to check if there is a top-level plan scheduled for the current time, if so, it discards those plans that are not scheduled for this time.

```
1 ...
2 [Info]:Current Top-Level Plan: managing-medications
3 [Info]:Current Plan Step (PS): at-living-room
4 [Info]:Checking time [at-living-room]
5 [Info]:PS [at-living-room] started at 7:15am
6 [Info]:PS [at-living-room] is running for 5 minutes
7 [Info]:PS [at-living-room] average time 3 minutes | tolerance 1 minute
8 [Warn]:PS [at-living-room] is taking too long
9 ...
```

Listing 1. Example of Failure Predictor Output.

The program output, presented in Listing 1, represents part of execution of the failure predictor approach, in a scenario where the user must take a medication in a strict time and during the plan execution the user's attention drawn to something else (e.g., the user stays at living room watching TV) and forgets to take his medication. In this case, the current top-level plan is *managing-medication* (Line 2) and the current plan step is *at-living-room* (Line 3). The failure predictor monitors the plan execution (Lines 4-6) and based on information in the `Plan-Step Controller`, i.e., average time of execution for each plan step and a time tolerance (Line 7), it is able to detect anomalies in plan execution and informs the system to try to address and correct these possible failures. In this example, the algorithm detects that a plan step is taking too long to be performed, then an alert message is generated (Line 8).

7. Conclusion

In this paper, we have developed a failure predictor based on plan recognition techniques and a calendar that includes *some* of the plans that the agent is known to be required to execute over time. Our main contribution is a system that anticipates plan failures by monitoring a sequence of agent actions during its plan execution. We have used this predictor as part of a system to support collaborative work in a scenario where family members and professional carers support an elderly person with a debilitating disease who lives alone. Although we currently only deal with plan failure *prediction*, as future work we can enable our system to elaborate alternative plans to avoid the detected failures (e.g., using planning techniques) rather than simply warning about possible plan failures.

Acknowledgements

Part of the results of this paper were obtained through the research project entitled "Semantic and Multi-Agent Technologies for Group Interaction", sponsored by Samsung Eletrônica da Amazônia Ltda. under the terms of Brazilian federal law No. 8.248/91.

References

Armentano, M. G. and Amandi, A. (2007). Plan recognition for interface agents. *Artificial Intelligence*, 28(2):131–162.

Avrahami-Zilberbrand, D. and Kaminka, G. A. (2005). Fast and complete symbolic plan recognition. In Kaelbling, L. P. and Saffiotti, A., editors, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 653–658. Professional Book Center.

Carberry, S. (2001). Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1-2):31–48.

Fox, M. and Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20(1):61–124.

Geib, C. W. (2002). Problems with intent recognition for elder care. In *Proceedings of the AAAI-02 Workshop Automation as Caregiver*, pages 13–17.

Geib, C. W. and Goldman, R. P. (2003). Recognizing plan/goal abandonment. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, pages 1515–1517, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Goldman, R. P., Geib, C. W., and Miller, C. A. (1999). A new model of plan recognition. *Artificial Intelligence*, 64:53–79.

Hong, J. (2001). Goal recognition through goal graph analysis. *Journal of Artificial Intelligence Research*, 15:1–30.

Kautz, H. A. and Allen, J. F. (1986). Generalized plan recognition. In Kehler, T., editor, *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI-86)*, pages 32–37. Morgan Kaufmann.

Masato, D. (2012). *Incremental Activity and Plan Recognition for Human Teams*. PhD thesis, University of Aberdeen.

Mausam and Weld, D. S. (2008). Planning with durative actions in stochastic domains. *Journal of Artificial Intelligence Research*, 31(1):33–82.

Panisson, A. R., Freitas, A., Schmidt, D., Hilgert, L., Meneguzzi, F., Vieira, R., and Bordini, R. H. (2015). Arguing About Task Reallocation Using Ontological Information in Multi-Agent Systems. In *12th International Workshop on Argumentation in Multiagent Systems*.

Pynadath, D. V. and Wellman, M. P. (1995). Accounting for context in plan recognition, with application to traffic monitoring. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 472–481, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Sukthankar, G., Goldman, R. P., Geib, C., Pynadath, D. V., and Bui, H. H., editors (2014). *Plan, Activity, and Intent Recognition: Theory and Practice*. Elsevier.