XXIII International Conference on Automated Planning and Scheduling

Application Showcase Proceedings

Rome, Italy - June 12, 2013

Edited By:

Nicola Policella and Nilufer Onder

Organizing Commitee

Nicola Policella
ESA-ESOC, Germany

Nilufer Onder
Michigan Technological University, USA

Preface

The Application Showcase Program is a forum that allows the ICAPS community to experience the latest contributions to the field. During the event, planning and scheduling researchers and practitioners demonstrate their state-of-the-art implementations in action, while the attendees have the opportunity to interact with the designers, developers, and their software.

The 2013 program includes ten demonstrations in the diverse areas of calendar management (Alexiadis and Refanidis), mobile target tracking (Bernardini et al.), plan visualization and management (Glinsky and Bartak), autonomous control (Munoz and R-Moreno), robotic bartending (Petrick and Foster), interactive storytelling (Porteous et al.), management of an electromagnetic surveillance space mission (Pralet et al.), scheduling and management of mining operations (Kameshwara et al.), maintenance of public infrastructures (Scharpff et al.), and network malware detection (Sohrabi et al.).

We express our gratitude to conference committee members and especially Conference Co-chair Simone Fratini and Publicity Co-chair Julie Porteous for supporting our work in organizing the Showcase. Special thanks go to IBM Research for sponsoring the best demo prize.

We would like to thank the authors for the remarkable work they performed. We invite you to explore and enjoy the impressive demonstrations.

Nicola & Nilufer

June 2013

Table of Contents

Post-Optimizing Individual Activity Plans through Local Search

Anastasios Alexiadis and Ioannis Refanidis
Department of Applied Informatics, University of Macedonia
Egnatia 156, 54006, Thessaloniki, Greece
talex@java.uom.gr, yrefanid@uom.gr

Abstract[1]

Post-optimization through local search is known to be a powerful approach for complex optimization problems. In this paper we tackle the problem of optimizing individual activity plans, i.e., plans that concern activities that one person has to accomplish independently of others, taking into account complex constraints and preferences. Recently, this problem has been addressed adequately using an adaptation of the Squeaky Wheel Optimization Framework (SWO). In this paper we demonstrate that further improvement can be achieved in the quality of the resulting plans, by coupling SWO with a post-optimization phase based on local search techniques. Particularly, we present a bundle of transformation methods to explore the neighborhood using either hill climbing or simulated annealing.We present several experiments that demonstrate an improvement on the utility of the produced plans, with respect to the seed solutions produced by SWO, of more than 6% on average, which in particular cases exceeds 20%. Of course, this improvement comes at the cost of extra time.

[1]The paper is published in the Proceedings of the workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS).

Autonomous Search and Tracking via Temporal Planning

Sara Bernardini, Maria Fox, Derek Long
Department of Informatics
King's College London
Strand London WC2R 2LS
name.surname@kcl.ac.uk

John Bookless
Advanced Technology Centre
BAE Systems
Bristol, UK, BS34 7QW
John.Bookless@baesystems.com

Abstract[1]

Search And Tracking (SAT) is the problem of searching for a mobile target and tracking it after it is found. As this problem has important applications in search-and-rescue and surveillance operations, recently there has been increasing interest in equipping unmanned aerial vehicles (UAVs) with autonomous SAT capabilities. State-of-the-art approaches to SAT rely on estimating the probability density function of the target's state and solving the search control problem in a greedy fashion over a short planning horizon (typically, a one-step lookahead). These techniques suffer high computational cost, making them unsuitable for complex problems. In this paper, we propose a novel approach to SAT, which allows us to handle big geographical areas, complex target motion models and long-term operations. Our solution is to track the target reactively while it is in view and to plan a recovery strategy that relocates the target every time it is lost, using a high-performing automated planning tool. The planning problem consists of deciding where to search and which search patterns to use in order to maximise the likelihood of recovering the target. We show experimental results demonstrating the potential of our approach.

[1]The paper is published in the Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013).

VisPlan – Interactive Visualisation and Verification of Plans

Radoslav Glinský, Roman Barták
Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
radkogl@gmail.com, bartak@ktiml.mff.cuni.cz

Introduction

Plan analysis is an inevitable part of complete planning systems. With the growing number of actions and causal relations in plan, this analysis becomes more and more complex and time-consuming process. In fact, plans with hundreds of actions are practically unreadable for humans. In order to make even larger plans transparent and human readable, we have developed a program that helps users with the analysis and visualization of plans. This program called VisPlan finds and displays causal relations between actions, it identifies possible flaws in plans (and thus verifies plans' correctness), it highlights the flaws found in the plan and finally, it allows users to interactively modify the plan and hence manually repair the flaws.

Existing Tools

Though the number of planners rapidly grows, the number of available tools for user interaction with planners is still limited. Two complex systems are worth mentioning as they are publicly available and provide graphical user interface supporting the planning process: itSimple (Vaquero et al. 2010) and GIPO (Simpson et al. 2007). They are both effective tools for modelling and updating planning domains, however, their plan analysis lacks some handy features such as:

- recognizing causal relations of actions,
- compact overview of actions' preconditions and effects
- support for plans with flaws,
- information about world state at a specific plan step,
- a user friendly interface to modify, insert, and delete actions in a plan and to re-verify the plan in real-time.

VisPlan focuses on all above features.

VisPlan

VisPlan is a graphical application (Figure 1) written in Java with the ultimate goal to visualize any plan, to find and highlight possible flaws, and to allow the user to repair these flaws by manual plan modification.

Program Input

VisPlan works with three types of files that the user should specify as program input:

- planning domain file in PDDL,
- planning problem file in PDDL,
- plan file specified either in text format or XML (eXtended Markup Language).

Currently, VisPlan supports STRIPS-like and temporal plans with propositional states. The program recognizes the plan types automatically and verifies and visualizes them in different ways. The following PDDL requirements are currently supported in the program: strips, typing, negative-preconditions, disjunctive-preconditions, equality, existential-preconditions, universal-preconditions, quantified-preconditions, durative-actions

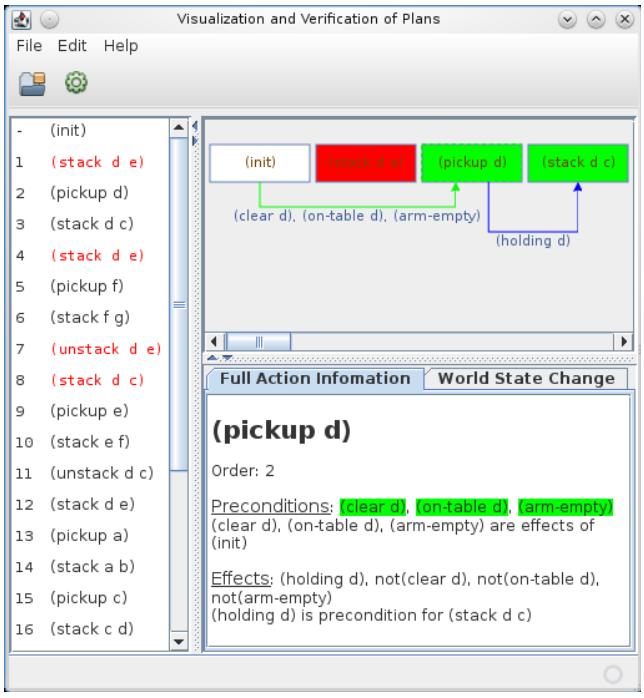

Figure 1. Graphical user interface of VisPlan.

Verification

Plan verification is automatically executed after the plan is initially loaded and then after each user interaction modifying the plan. Plan verification runs in a separate program thread, thus ensuring that the main GUI responds to user's actions even if verification takes a long time. The verification process is based on simulation of plan execution. Firstly, it constructs an initial state of the world. After that, it consecutively tries to apply a single action (in the order given by the sequential plan) to the current world state. If the action is applicable, the action is applied and a new world state is computed based on effects of the action. If the action is not applicable, its effects are not encountered and the program starts processing the next action of the plan. Causal relations to previous actions in the plan are computed for each action during its verification.

In case of durative actions in temporal plans, the world state may be changed more than once, when both "at start" effects and "at end" effects of the actions are encountered. Furthermore, a single durative action needs to be checked multiple times. Besides "at start" conditions (equivalent to STRIPS preconditions), an action can be generally defined with the "over all" and with the "at end" conditions.

The verification process creates new virtual actions representing starts and ends of real actions. These actions are then examined in a given temporal order. In case the end time of an action is equal to the start time of the next action, the virtual action representing the end time is examined before the start of the next action.

At the first examination of an action, at its start time, the action is applied (considering its "at start" effects) provided that its "at start" conditions are satisfied. In addition, the action is marked to be "in progress". If an action has been applied at its start time, it will be examined at its end time (start time plus duration), too. Similarly, the action's "at end" effects are applied if its "at end" conditions are satisfied and the "in progress" tag is removed.

Besides checking action's own conditions, the algorithm also verifies satisfiability of the "over all" conditions of the actions being "in progress". Such verification is performed only when an inducing action is applied (either at the start time or at the end time of the inducing action).

When applying effects of actions in temporal plans, the verification process also takes special care to ensure that:

- effects applied by any two actions at the exactly same time must be mutex-free
- effects of an action cannot be used at the exactly same time as conditions for other actions

If any of the two above requirements fails, the latter of the two involving actions is marked as mutex-containing and cannot be applied. Afterwards, the action is visualized differently. The reason why the latter action has been marked as mutex-containing and not the one, which had induced the mutex (or both actions), is the unique concept of VisPlan regarding plans' manipulation. Generally, the verification process continues even if an invalid or non-applicable action is found. Such actions are simply omitted. In the case when an action's "at start" effects have been applied at action's start time and it has been found later that any of the action's "over all" or "at end" conditions cannot be satisfied, the verification process is rolled back to the point when the affected action was applied at its start time, the action is then omitted and marked as non-applicable.

Visualization

As shown in Figure 2, plan's actions are visualized as boxes of fixed size filled by action name. Each action is coloured either green or red depending on whether the action is applicable or non-applicable. Causal relations between the actions are visualized by edges. These edges are annotated by grounded facts that are "passed" between the actions. Only the causal relations for the currently highlighted action are displayed to remove cluttered view.

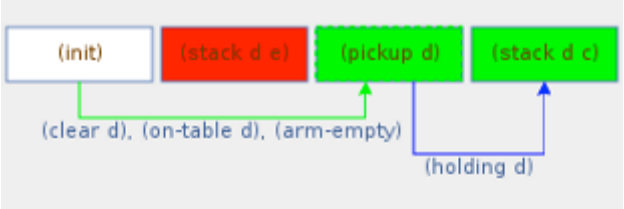

Figure 2. Highlighted causal relations in plan.

There are two special "actions" displayed in the graph to represent the initial state and the goal. A classical plan-space approach is used to define these actions. The init action has empty preconditions and the facts that apply at the initial state are considered as its effects. The goal action has empty effects and the set of facts that need to be satisfied at the final world state are considered as its preconditions. This way we are able to visualize causal relations also at the margins of the plan, thus showing dependencies on the initial state, final actions giving the goal conditions, and possibly missing goal conditions.

The major extension of temporal plans is that actions have duration and may overlap in time. We include this information in visualisation – the boxes have the length dependent on the action duration and if two actions overlap in time, the action starting later is displayed in the first available row (Figure 3).

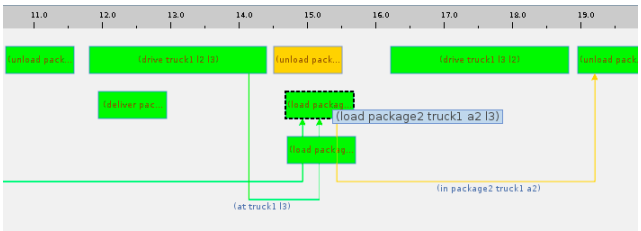

Figure 3. Visualization of a temporal plan.

For the highlighted action, the system displays complete information about the action including satisfied and violated preconditions and actions giving these preconditions (Figure 4). Alternatively the user may ask to show the state as sets of valid predicates before and after the action.

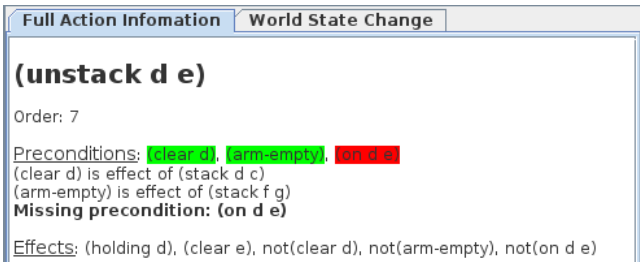

Figure 4. Complete information about the selected action and its role in the plan.

Plan Modifications

In addition to visualization of plans the software supports interactive modification of the plans. The following operations with plans are supported:

- inserting new actions (selection of actions and their parameters is automatically restricted to the current planning domain and problem),
- removing actions,
- modifying actions (changing the parameter of the action so a different action based on the same operator can be obtained),
- changing the order of actions (for temporal plans this means changing action allocation in time).

After each modification the plan is immediately re-verified. For example, while dragging an action (to change its position), the plan is automatically re-verified to immediately display the modified causal relations by highlighting the actions providing preconditions and actions using effects of the action being dragged. Together with the information about the new satisfiability of the actions (including the goal action), the user instantly knows where he/she can drop off the action. In STRIPS-like plans, the actions are just swapped with each other while dragging. In temporal plans, the positioned action is automatically vertically adjusted so that the actions do not graphically overlap.

As any aspect of the plan can be modified, the user can use the system to manually build plans while being informed about non-satisfied goals and preconditions. All the modifications are revertible both backward and forward. Modified plans can be saved in the text format to either the same (initially loaded) file or to a new file.

Summary

VisPlan is a system for visualizing classical and temporal plans in the Gantt style. One of it is important features is showing causal relations between the actions so it is easy to verify if an action pre-condition or a goal condition are not met. The second important feature is support for manual modification of plans so the system can also be used for manual planning. The up-to-date version of the software is available at http://glinsky.org/visplan. The program is under development and several additional features are assumed in the next versions. In particular, the following extensions are under development:

- wider PDDL support,
- own planning module,
- support for finding possible plan modifications in order to solve flaws in the plan (the system suggests what to do to repair a particular flaw for example by adding a new action),
- graphs visualizing a timeline of predicates and numerical variables during plan execution.

Acknowledgements

The research is supported by the Czech Science Foundation under the contract P103/10/1287.

References

Simpson, R.M.; Kitchin D.E.; McCluskey, T.L. 2007. Planning Domain Definition using GIPO. *The Knowledge Engineering Review* 22(2): 117-134.

Vaquero, T. S.; Silva, J. R.; Beck, J.C. 2010. Analyzing Plans and Planners in itSIMPLE3.1. In: Proceeding of the ICAPS 2010 Knowledge Engineering for Planning and Scheduling Workshop. Toronto. Canada, pp. 45-52.

Model-Based Architecture on the ESA 3DROV simulator

Pablo Muñoz and María D. R-Moreno
Departamento de Automática, Universidad de Alcalá
Ctra. Madrid-Barcelona Km 33,600 E-28871
Alcalá de Henares, Madrid

Abstract

We have developed the MoBAr architecture, a general 3 layer (3T) architecture based on models, focused on building a flexible, adaptable and reusable autonomous control architecture. It follows an incremental approach design philosophy, which implies that a basic model can be quickly deployed and later refined to add new functionality.
The deliberative layer of the architecture integrates task-planning and path-planning using an interleaving schema that allows us to obtain better paths using a Digital Terrain Model (DTM) of the terrain and traversal costs. The executor implements decomposition of high-level actions into functional layer commands, monitor the general state of the robot and world, and reactive behaviors to react into a dynamical environment. The functional layer is responsible of safe access to the hardware of the robot and providing its functionality to the upper layers.

Model Based Architecture

The MoBAr architecture developed in the University of Alcalá (UAH) corresponds to a three layers (3T) system (Gat 1998), in which the top tier will be in charge of the deliberative process, long-term memory and learning process as a function of events that occur in the environment. The middle level or execution system also has a short-term memory, as well as a series of rules that trigger the reactive behavior implemented, in order to response in a short time to eventual situations that may occur in both, the environment and the internal state of the robot. Finally, the low level or functional level, is responsible of providing the functionality of the robot, and relay the information collected by the sensors.

Each layer is based on a model with different levels of abstraction. A model defines the properties, capabilities and constraints of the robot at each layer. Starting from the functional layer, which represents the hardware abstraction level, it contains the definition of the internal state of the robot plus the abilities that it has. The upper layers have fewer detailed models, and thus, less coupled with the underlying hardware. In this way, the executor is in charge of taking high level actions coming from the deliberator and decomposing them into lower level commands supported by the functional layer. So, we want that the executor model has little relationship with the hardware, and the high level model only knows the hardware in terms of goal oriented actions and high level abstraction of constraints.

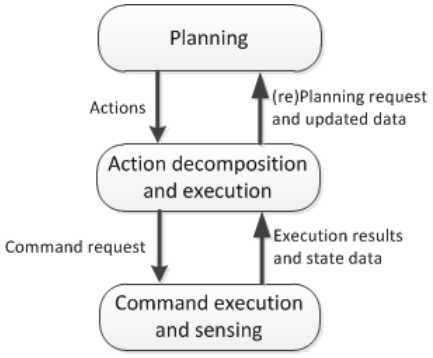

Figure 1: Execution cycle in MoBAr.

To implement the MoBAr architecture we have taken advantage of different general purpose technologies: for the deliberative layer we have used the PDDL language (McDermott 1998) and a PDDL-based planner. We have chosen the Universal Executive and its language, PLEXIL (PLan EXecution Interchange Language) (Verma et al. 2006) to model and control the executor, and, finally the GenoM2 (Generator Of Modules) framework (Mallet, Fleury, and Bruyninckx 2002) for the definition and implementation of the functional layer. The first two layers, deliberator and executor, use models represented by a language that will be interpreted by a program, and thus, are easily interchangeable in order to adapt them to multiple robots. Instead, the functional layer is strongly dependent on the hardware.

The architecture follows a sense-(re)plan-act cycle (see fig. 1): at first time, the system takes the initial state of the world (state from the functional layer and the PDDL problem), the goals to achieve and obtains a feasible plan. This plan is static: the executor takes each action, decompose it in a set of commands and send it to the functional layer. With the results of these commands execution, the executor checks if there is an unexpected situation, and, if it cannot handle that situation, update the information of the world, goals currently achieved and state of the robot, and then the planner must obtain a new plan.

Focusing on the deliberative capabilities, PDDL-based

planners are systems that use two input files to represent their knowledge base. One of the files contains a description of the actions that represent "what can/cannot be done" and the other file includes the three elements which define the problem: the known objects of the world, the initial state and the goals we want to achieve. With this information, the planner searches a sequence of actions that can reach the goals from the initial state. The optimal solution of the problem is a conjunction of two factors: the metric used and the resolution algorithm.

Currently we are using SGplan$_6$ (Hsu and Wah 2008) a PDDL-planner that accepts the PDDL version 2.1 (Fox and Long 2003) and common features of PDDL 3 (Gerevini and Long 2005). With PDDL 2.1 we can determinate how long each action will take and, using fluents, we can establish a basic resource model for the energy consumption of the actions, and consistently decide whether a plan is feasible or not in terms of total amount of energy consumed. Also, using the version 3, we can employ goals as preferences, so if there is an unreachable goal, we can obtain a plan that ignores it.

Besides, for the path-planning problem, that is, the route to follow between scientific tasks, we have implemented an interleaving schema between the PDDL-based planner, the task-planner, and an algorithm for path-planning. Using a greedy path-planning algorithm (Muñoz and R-Moreno 2012) we compute a suboptimal path-cost between each pair of points to reach and provide that information for the task-planner in order to try to optimize the distance and time spent in the movements. When a plan is obtained, the movement actions in the plan are replaced by specific routes generated by a different path-planning algorithm that employs the DTM data.

Problem definition

To test the architecture, we define a typical scenario for an exploration rover. The scenario, represented in fig. 3, consists of the operation of the ExoMars to achieve the acquisition of two pictures in different locations (and possibly with different pointing of the pan-tilt unit), a drill in one of these locations and then, return to the start point. Also, during the traverse there could exist visibility windows in which the rover can transmit the data acquired to a station. The objective is to send all the possible pictures taken during these windows.

To safely operate the rover, there is a little set of constraints that must be included in the models:

- The rover is able to move between two points in space given their coordinates (x, y).
- The rover has different navigation speeds. The functional layer could provide a signal to identify difficult terrain in which is required to move in low speed for safety reasons.
- The pan-tilt unit can be moved to aim a desired point, given by their angles (α, β).
- During all the acquisition of a picture, the pant-tilt unit must be pointing at the desired site and the rover must stay stopped.

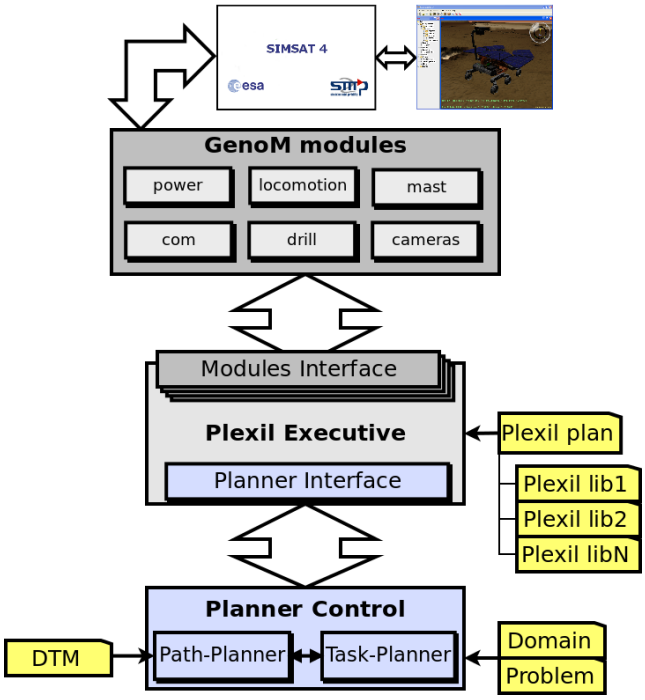

Figure 2: Model-Based Architecture design for ExoMars.

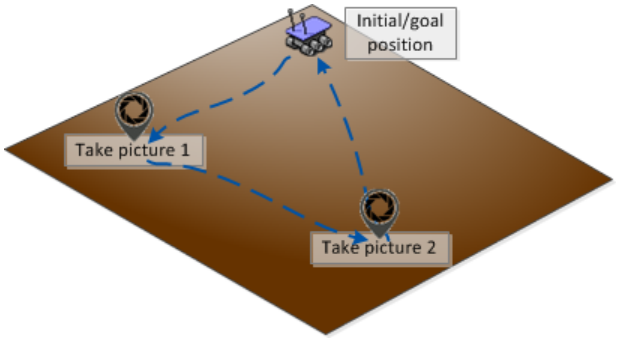

Figure 3: Problem representation.

- To perform a drill operation, the rover must stay still.
- When the rover is moving, the pan-tilt unit must be pointing to the front, that is, $(0, 0)$.
- It is possible to transmit pictures while the rover is stopped and there is a visibility window.

Deliberative and executive models

The high layer or deliberator of the architecture has the function to obtain a feasible and safely route between the initial position of the rover and a final point, performing the defined science objectives during the planned trajectory. In order to describe this layer, we differentiate five elements: (i) the task-planner, SGplan$_6$; (ii) the PDDL files which contain the models for the rover and terrain, and the actions that the rover can perform (both, input files to the planner); (iii) the path-planner, in charge of obtaining a feasible and safe path;

(iv) the DTM and cost map definition of the environment, and (v) a library that manages these elements and provides a planning framework to the executor.

Both, the task-planner and path-planner can be dynamically replaced in execution time. The planner election is conditioned by the algorithm resolution and the PDDL version that it supports. We need a planner that can manage metrics, fluents and constraints (that is, PDDL version 2 and some elements of version 3). The path-planner depends on the terrain, for example, for flat terrains there is no need to employ costly searches over a DTM model and it is possible to use a classical path-planning algorithm.

The problem and domain files contain the knowledge of the rover and the actions that it can perform. The domain specifies the actions, such as go ahead, rotate, drill, recharge or acquire images. These actions are high level actions and the executor is on charge of decomposing them into a valid sequence of commands that the functional layer can execute. The actions representation include the duration and the energy consumption. For both elements, the locomotion duration and consumption is based on the rover speed and the traveled distance. The other actions have a fixed cost specified in the problem file. The energy is treated as a fluent, but it is not the best solution; the energy model can be more effective if it were treated as a resource (some planners that we try do not find a solution when the energy is not enough to achieve goals and the rover needs to recharge).

The terrain model is defined in the DTM and cost files. The DTM contains the altitude of each point of the map and the cost file defines the obstacles and the traversal cost to move through every region. This allows different path-planning strategies to find paths: focusing on the altitude difference, avoiding to cross regions with high traversal costs (potentially dangerous) or combining them.

The problem file moreover, includes the model that defines the rover position, available subsystems and associated data, and the goals that we want to achieve. The ExoMars PDDL model is shown in Fig. 4. The rover model contains the initial position and orientation, and the subsystem definitions. Each subsystem is defined by a predicate to indicate that this subsystem is available, and a set of functions that define the energy consumption and operations time of the subsystems. For example, the locomotion subsystem has different speeds with different energy consumption in function of the speed. The last element of the problem is the goal(s) definition. It defines the tasks that the rover must perform, such as, go to a desired location, drill in a specific point or take an image from a location and with a particular pan tilt pointing and mode. If the planner supports plan preferences, one or more targets can be not satisfied by the planner in function of a penalty defined for each goal. This implies that if a goal is too expensive to achieve and its penalty is low, the planner is allowed to skip this target in the plan.

The executive is the coordinator of the other two layers. The executor is connected with both, the functional layer and the deliberator, and it is responsible of coordinating them. In addition to the coordination functionality, the executor integrates reactive skills and fail-safe routines.

The model that guides the execution flow is a set of hierarchical plans written in PLEXIL. The top PLEXIL plan is responsible of the planning/replanning processes, and the interaction between the executor and the planner control. This is associated with the high level interface adapter which allows to read the plan obtained by the deliberator and access and modify the information contained in the problem file.

```
(define (problem appshow)
(:domain MoBAr-exomars)
(:objects
   C0_0 C1_0 C2_0 C2_1 - loc
   NavCam - cam lowRes - mode
   plat0_0 plat35_45 - platpos
   hispeed lowspeed - navmode
   exomars - rover
)
(:init
 (= (time_point) 0)
;LOCATION AND DISTANCES
 (= (distance_to_move C0_0 C2_0) 2) ; m
 (= (distance_to_move C0_0 C2_1) 2.24)
 (= (distance_to_move C2_0 C0_0) 2)
 (= (distance_to_move C2_0 C2_1) 1)
 (= (distance_to_move C2_1 C0_0) 2.24)
 (= (distance_to_move C2_1 C2_0) 1)
;EXOMARS ROVER DATA AND CONFIG
 (position exomars C0_0)
 (= (energy exomars) 2.400) ; A
 (= (energy_cons exomars) 0)
 (has_locomotion exomars)
  (navigation_mode exomars hispeed)
  (= (speed exomars hispeed) 0.25) ; m/seg
  (= (speed exomars lowspeed) 0.1) ; A/m
  (= (power_per_dis exomars hispeed) 0.008)
  (= (power_per_dis exomars lowspeed)0.006)
 (camera_mode exomars NavCam lowRes)
  (= (transmit_energy exomars) 0.03)
  (= (camera_energy NavCam lowRes) 0.001)
  (= (time_to_picture NavCam lowRes) 5)
  (= (start_win C1_0) 20)
  (= (end_win C1_0) 35)
  (= (start_win C2_1) 70)
  (= (end_win C2_1) 90)
 (platine_pos exomars NavCam plat0_0)
  (= (t_move_platine plat0_0 plat35_45) 10)
  (= (t_move_platine plat35_45 plat0_0) 7)
  (= (platine_energy) 0.001)
 (has_drill exomars)
 (= (drill_energy) 0.008)
  (= (time_to_drill C2_0) 20)
)
(:goal (and
 (position exomars C0_0)
 (transmited C2_0 lowRes plat35_45)
 (transmited C2_1 lowRes plat35_45)
 (drilled C2_0)
 )
)
(:metric minimize (+ (total-time)
            (*100 (energy_cons exomars))) )
)
```

Figure 4: PDDL definition for the ExoMars example.

When the plan is valid, the Plexil Executive reads the actions one by one, and executes them. Each action must be associated with a PLEXIL node (normally is a PLEXIL library, that is, an external PLEXIL plan) that manages the correct decomposition and execution of the action.

For each possible action, there is a G^{en}oM module that executes it. Examples of possible actions obtained from the planner are rotate, drill or recharge. The rotate action corresponds to a rotation request of the Locomotion module. The drill action is responsibility of the Drill module. Recharge is managed by the Power module. In order to connect each G^{en}oM module with the executor, there is a interface adapter that communicates the module with the Plexil Executive. This interface sends and monitors the requests to the G^{en}oM module, and catches the result sent by the module. If the execution is correct, the executor continues its plan without change, but if an error is reported, the PLEXIL plan is responsible of finding a solution. For example, if a minor camera failure is detected, the PLEXIL plan can skip this image acquisition, or try to change the camera mode. For locomotion problems, such as trying to cross through more complicated terrain than expected, the PLEXIL plan can stop the motion and propagate new terrain data to the deliberator in order to try to obtain a new route. When problems are more serious, the defined behavior in the PLEXIL plan must be to set the rover into a safe state or to wait for human intervention.

Each PLEXIL plan that carries out the execution of an action can also contains PLEXIL libraries (if needed). This allows us to expand the functionality of the executor without modifying anything else in the architecture.

Conclusion and future work

In this paper we have presented an initial version of MoBAr adapted to the ExoMars rover. The design philosophy used (based on general purpose systems), has allowed us to adapt and improve the models of each layer according to the necessities of the rover.

Since both, the high level layer and the executor are made by general purpose systems, this work can be focused on the study of the model design of each layer. However, an important conclusion we have reached is that the models of each layer are strongly dependent on the attached layers, that is specially significant for the power model. This involves some problems due to the different vision of the world that each layer has. In our case, the G^{en}oM module is in charge that the power subsystem has a power model based of the instant consumption of each subsystem power status (off, standby, heating, etc.), and that the PDDL model has a less precision model that manages full operations (drill, move, etc.). Between them the executor must deal with both representations in order to detect and correct inconsistencies on the power consumption during execution.

There is still work to be done to provide full autonomy to the architecture, such as implementing different reactive behaviors in the PLEXIL plans to support subsystem monitoring, better control of the resources and plan adaptation. Also, some engineering effort must be done in the functional modules to provide required functionality such as stereo vision to correct the DTM and also, to provide support for opportunistic science discovery.

Therefore, we believe that this architecture can be a starting point for a more complex control system that will be based on the design of high level models rather than on the underlying implementation issues.

References

Fox, M., and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *AI Research* 20:61–124.

Gat, E. 1998. Three-layer architectures. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., eds., *Mobile Robots and Artificial Intelligence*, 195–210. AAAI Press.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. In *The Language of the Fifth International Planning Competition*.

Hsu, C., and Wah, B. 2008. The SGPlan planning system in IPC-6. In *Sixth International Planning Competition*.

Mallet, A.; Fleury, S.; and Bruyninckx, H. 2002. A specification of generic robotics software components: future evolutions of GenoM in the orocos context. In *International Conference on Intelligent Robotics and Systems*.

McDermott, D. 1998. The PDDL planning domain definition language. *The AIPS-98 Planning Competition Comitee*.

Muñoz, P., and R-Moreno, M. D. 2012. Improving efficiency in any-angle path-planning algorithms. In *6th IEEE International Conference on Intelligent Systems (IEEE-IS)*, 213–218.

Verma, V.; Jnsson, A.; Pasareanu, C.; and Iatauro, M. 2006. Universal Executive and PLEXIL: Engine and language for robust spacecraft control and operations. In *American Institute of Aeronautics and Astronautics Space Conference*.

Plan-Based Social Interaction with a Robot Bartender

Ronald P. A. Petrick
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
rpetrick@inf.ed.ac.uk

Mary Ellen Foster
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh EH14 4AS, Scotland, UK
M.E.Foster@hw.ac.uk

Abstract

A robot coexisting with humans must not only be able to perform physical tasks, but must also be able to interact with humans in a socially appropriate manner. We describe an application of planning to task-based social interaction using a robot that must interact with multiple human agents in a simple bartending domain. The resulting system infers social states from low-level sensors, using vision and speech as input modalities, and uses the knowledge-level PKS planner to construct plans with task, dialogue, and social actions.

Introduction and Motivation

As robots become integrated into daily life, they must increasingly deal with situations in which *socially appropriate interaction* is vital. In such settings, it is not enough for a robot simply to achieve its task-based goals; instead, it must also be able to satisfy the social goals and obligations that arise through interactions with people in real-world settings. As a result, a robot not only requires the necessary physical skills to perform objective tasks in the world, but also the appropriate *social skills* to understand and respond to the intentions, desires, and affective states of its interaction partners. To address this challenge, we are investigating *task-based social interaction* in a bartending domain, by developing a robot bartender (Figure 1) that is capable of dealing with multiple human customers in a drink-ordering scenario.

Key to our approach is the use of high-level planning techniques, which are responsible for action selection and reasoning in the robot system. Specifically, we use the knowledge-level planner PKS (Petrick and Bacchus 2002; 2004), a choice that is motivated by PKS's ability to work with incomplete information and sensing actions: not only must the robot perform physical tasks (e.g., handing a customer a drink), it will often have to gather information it does not possess from its environment (e.g., asking a customer for a drink order). Moreover, since interactions will involve human customers, speech will be the main input modality and many of the planner's actions will correspond to speech acts, providing a link to natural language processing—a research field with a long tradition of using planning, but where general-purpose planning techniques are not the focus of mainstream study.

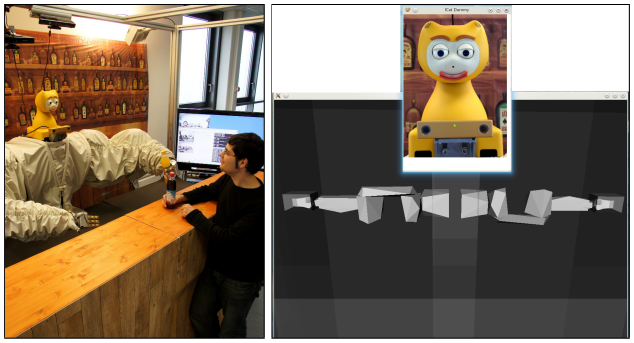

Figure 1: The JAMES robot bartender

While planning offers a tool for action selection, it is only one component in a larger system that operates in a real-world environment. A second, central component in our system is the state manager which mediates between the low-level input sensors and the planner, and which overcomes some of the representational difficulties involved in bridging the gap between continuous, low-level input streams and symbolic, high-level state-based reasoning.

In the rest of the paper we give a technical description of the robot system, with a focus on the role of the planner and how it is integrated in this framework. The application of this work is a simple bartending scenario, which is modelled as a PKS planning domain. More details on this work can be found in (Petrick and Foster 2013). This work forms part of a project called JAMES (Joint Action for Multimodal Embodied Social Systems; see `james-project.eu`).

Robot System Architecture and Components

The target application for this work is a bartending scenario, using the robot platforms shown in Figure 1. The robot hardware itself (Figure 1) consists of two 6-degrees-of-freedom industrial manipulator arms with grippers, mounted to resemble human arms. Sitting on the main robot torso is an animatronic talking head capable of producing facial expressions, rigid head motion, and lip-synchronised synthesised speech. For testing and demonstration purposes, the simulated robot shown in Figure 1 is also available.

A sample interaction in a simple bartending scenario is shown in Figure 2. In this example, two customers enter the bar and attempt to order a drink from the bartender. When

A customer approaches the bar and looks at the bartender	
ROBOT:	[Looks at Customer 1] How can I help you?
CUSTOMER 1:	A pint of cider, please.
Another customer approaches the bar and looks at the bartender	
ROBOT:	[Looks at Customer 2] One moment, please.
ROBOT:	[Serves Customer 1]
ROBOT:	[Looks at Customer 2]
	Thanks for waiting. How can I help you?
CUSTOMER 2:	I'd like a pint of beer.
ROBOT:	[Serves Customer 2]

Figure 2: An example interaction in the bartending scenario.

the second customer appears while the bartender is engaged with the first customer, the bartender reacts appropriately by telling the second customer to wait, finishing the current transaction, and then serving the second customer.

Even this simple interaction presents challenges which have motivated the design of the overall system: a vision system must track the locations and body postures of the agents; a speech-recognition system must detect and deal with speech in an open setting; reasoning components must determine that both customers require attention and ensure they are served in the correct order; while the output components must select and execute concrete actions for each output channel that correctly realise high-level plans. The software architecture of the robot system is shown in Figure 3, with the main components highlighted below.

Input Processing: One of the primary input channels for the robot is computer vision. The full JAMES vision system tracks the location, facial expressions, gaze behaviour, and body language of all people in the scene in real time, using a set of visual sensors (Baltzakis, Pateraki, and Trahanias 2012); a limited-functionality vision system is also available that can run on a single Kinect for demo and testing purposes. Information from the vision system is constantly published to the state manager multiple times a second.

The other primary input modality in the system is linguistic, combining a speech recogniser with a natural-language parser to create symbolic representations of the speech produced by all users. For speech recognition, we use the Microsoft Kinect and the Microsoft Speech API, with a scenario-specific speech grammar to constrain the recognition task. Recognised speech is then parsed using a grammar implemented in OpenCCG (White 2006); the grammar contains syntactic and semantic information, and is used both for parsing the spoken input and for surface realisation of the selected output (see below). The parsed speech, confidence score, and source angle are passed to the state manager.

State Management: The primary role of the state manager is to turn the continuous stream of messages produced by the low-level input components into a discrete representation that combines social and task-based properties. The state representation is based on a set of *fluents*: first-order predicates and functions that denote particular qualities of the world, the robot, and other entities in the domain. A state is a snapshot of all fluent values at a given point in time. Intuitively, states represent a point of intersection between

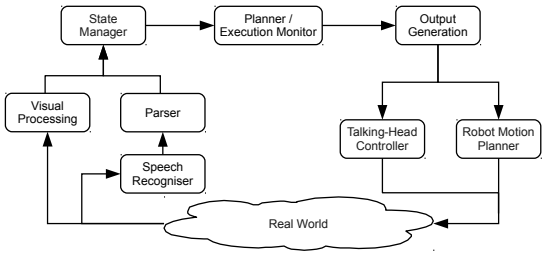

Figure 3: Software architecture of the robot system

low-level sensor data and the high-level structures used by components like the planner. Since states are induced from the mapping of sensor observations to fluent values, the challenge of building an effective state manager rests on defining appropriate mapping functions.

In the bartender robot, we treat each low-level input component as a set of sensors. The linguistic interpreter corresponds to three sensors: two that observe the parsed content of a user's utterance and its associated confidence score, and another that returns the estimated angle of the sound source. The vision system also senses a large number of properties about the agents and objects in the world, each of which corresponds to a set of individual sensors. Certain low-level output components are also treated as sensors. For example, the robot arms provide information about the start and end of manipulation actions, while the speech synthesiser reports the start and end of all system utterances. Modelling output components as sensors allows information from these sources to be included in the derived state, ensuring the current state of interaction is accurately reflected (e.g., the state of turn-taking or the completion of physical actions).

In the current robot bartender system, the state includes information about all agents in the scene: their locations, torso orientations, attentional states, and drink requests if they have made one. The mapping from sensors to states is rule-based. One set of rules infers user social states (e.g., seeking attention) from the low-level sensor data, using guidelines derived from a study of natural bartender interactions (Huth 2011). The state manager also incorporates rules that convert the logical forms produced by the parser into communicative acts (e.g., drink orders), and that use the source angle from the speech recogniser together with the vision properties to determine which customer is likely to be speaking. A final set of rules determines when new state reports are published, which controls turn-taking.

To deal with the more complex states required in future versions of the bartender system, we are currently exploring the use of supervised learning classifiers trained on multimodal corpora. In an initial study, the trained classifiers significantly outperformed the hand-coded rules both in cross-validation and when tested with real users (Foster 2013).

Planning and Execution Monitoring: The high-level planner is responsible for taking state reports from the state manager and choosing actions to be executed on the robot. Plans are generated using PKS (Planning with Knowledge and Sensing) (Petrick and Bacchus 2002; 2004), a conditional planner that works with incomplete information and

sensing actions. PKS operates at the *knowledge level* and reasons about how its knowledge state, rather than the world state, changes due to action. To do this, PKS works with a restricted first-order representation with limited inference. While features such as functions and run-time variables are supported, these restrictions mean that some types of knowledge (e.g., general disjunctive information) cannot be modelled. To ensure efficient inference, PKS restricts the type of knowledge it can represent to a set of four databases:

$\mathbf{K_f}$: This database is like a STRIPS database except that both positive and negative facts are permitted and the closed world assumption is not applied. K_f can include any ground literal or function (in)equality mapping ℓ, where $\ell \in K_f$ means "the planner knows ℓ."

$\mathbf{K_w}$: This database models the plan-time effects of "binary" sensing actions. $\phi \in K_w$ means that at plan time the planner either "knows ϕ or knows $\neg\phi$," and that at run time this disjunction will be resolved. PKS uses such information to build conditional branches into a plan.

$\mathbf{K_v}$: This database stores functions whose values will become known at run time. In particular, K_v can model the plan-time effects of sensing actions that return terms. K_v can contain any unnested function, where $f \in K_v$ means that at plan time the planner "knows the value of f."

$\mathbf{K_x}$: This database models the planner's "exclusive-or" knowledge. Entries in K_x have the form $(\ell_1|\ell_2|\ldots|\ell_n)$, where each ℓ_i is a ground literal. Such formulae represent a type of disjunctive knowledge common in planning domains, namely that "exactly one of the ℓ_i is true."

A PKS action is modelled by a set of *preconditions* that query PKS's knowledge state, and a set of *effects* that update the state. Preconditions are a list of simple questions about PKS's knowledge state (e.g., a query $K(\phi)$ asks if ϕ is known). Effects are described by a set of STRIPS-style "add" and "delete" operations that modify the contents of individual databases. E.g., *add*(K_f, ϕ) adds ϕ to the K_f database, while *del*(K_w, ϕ) removes ϕ from K_w. PKS constructs plans by reasoning about actions in a simple forward-chaining manner, and can build plans with branches be considering the possible outcomes of its K_w and K_v knowledge. *Goals* are specified in a form similar to action preconditions.

PKS is also aided by an execution monitor which controls replanning. The monitor takes as input a PKS plan, and a description of the sensed state provided by the state manager. The monitor must assess how close an expected, planned state is to a sensed state in order to determine whether the current plan should continue to be executed. To do this, it tries to ensure that a state still permits the next action (or set of actions) in the plan to be executed, by testing an action's preconditions against the current set of (sensed) state properties. In the case of a mismatch, the planner is directed to build a new plan, using the sensed state as its initial state.

Output Generation: Output in the system is based on dividing actions selected by the planner into speech, head motions, and arm manipulation behaviours that can be executed by the robot. To do so, we use a structure containing specifications for each of the output modalities (Isard and Matheson 2012), based on a rule-based approach which splits each planned action into its component subparts. The resulting structure is then passed to the multimodal output generator, which sends specific commands to each output channel.

action *ask-drink(?a : agent)*
preconds: *K(inTrans = ?a)* $\wedge$ $\neg$*K(ordered(?a))* $\wedge$ $\neg$*K(otherAttnReq)* $\wedge$ $\neg$*K(badASR(?a))*
effects: *add(K_f, ordered(?a)), add(K_v, request(?a))*

action *serve(?a : agent, ?d : drink)*
preconds: *K(inTrans = ?a)* $\wedge$ *K(ordered(?a))* $\wedge$ *K_v(request(?a))* $\wedge$ *K(request(?a) = ?d)* $\wedge$ *K(otherAttnReq(?a))* $\wedge$ $\neg$*K(badASR)* $\wedge$ $\neg$*K(ackOrder(?a))*
effects: *add(K_f, served(?a))*

Figure 4: Example PKS actions in the bartender domain

OpenCCG is used to generate speech output for the robot, using the same grammar that is used to parse the input. The output description is specified in terms of high-level communicative acts, which are translated into logical forms and sent to the OpenCCG realiser. The realiser then outputs text strings that are turned into speech by the robot's animatronic head. In addition to speech, the robot also expresses itself through facial expressions, gaze, and arm manipulation actions. The animatronic head can produce a number of expressions and can gaze at customers or objects, while the robot arm can perform tasks like grasping to hand over a drink to a customer; motion planning and robot control make use of the Robotics Library (Rickert 2011).

System Integration: Like most interactive multimodal systems, the robot bartender is made up of a number of distributed, heterogeneous software components, drawing on diverse research paradigms, each with individual hardware and software requirements. These components must all communicate with one another to support interactions in the bartender scenario. The planner must also be situated in this system and use the same interfaces as other components.

For inter-module communication in the robot bartender, we use the Ice object middleware (Henning 2004), which provides platform- and language-independent communication among the modules and supports direct module-to-module communication as well as publish-subscribe messaging. On the planning side, adapting the off-the-shelf PKS planner for use with Ice is achieved by creating a communication-level API to common planning features, and re-engineering the backend planner into a suitable library that supported this interface. Common operations like planner configuration, domain definition, and plan construction were abstracted into a class definition that allowed a PKS planner instance to be created as a C++ object. The interface to this library was built into a simple server which provided a transparent network interface to its functions over Ice.

Planning Interactions for Social Behaviour

The robot's available high-level actions are modelled as part of a PKS planning domain, rather than using specialised tools as is common in many dialogue systems. For instance,

the basic bartender domain consists of the following actions, available to the robot for interacting with human customers:

greet(?a)	greet an agent *?a*,
ask-drink(?a)	ask agent *?a* for a drink order,
ack-order(?a)	acknowledge agent *?a*'s drink order,
serve(?a, ?d)	serve drink *?d* to agent *?a*,
bye(?a)	end an interaction with agent *?a*,
not-understand(?a)	inform agent *?a* was not understood,
wait(?a)	tell agent *?a* to wait, and
ack-wait(?a)	thank agent *?a* for waiting.

Actions model high-level robot behaviours that include a mix of physical, sensory, and speech acts. Examples of two PKS actions in the bartender domain are shown in Figure 4.

Information about human agents is not hard-coded in the domain but is detected by the vision system and passed to the planner by the state manager through its state updates. Similarly, changes to the agent list are also sent to the planner in state reports, causing it to update its domain model. The goal is simply to serve each agent seeking attention. This goal is viewed as a rolling target which is reassessed each time a state report is received by the planner. For instance, if two agents (*a1* and *a2*) are seeking attention, PKS can build the following plan (similar to the interaction in Figure 2):

wait(a2),	[Tell agent *a2* to wait]
greet(a1),	[Greet agent *a1*]
ask-drink(a1),	[Ask *a1* for drink order]
ack-order(a1),	[Acknowledge *a1*'s drink order]
serve(a1, request(a1)),	[Give the drink to *a1*]
bye(a1),	[End *a1*'s transaction]
ack-wait(a2),	[Thank *a2* for waiting]
ask-drink(a2),	[Ask *a2* for drink order]
ack-order(a1),	[Acknowledge *a2*'s drink order]
serve(a2, request(a2)),	[Give the drink to *a2*]
bye(a2).	[End *a2*'s transaction]

Here, *a1*'s drink order is taken and processed, followed by *a2*'s order. The *ask-drink* action is a sensing action that returns information about the term *request* (an agent's drink order), which is then used as a run-time variable in the *serve* action. The *wait* and *ack-wait* actions are used to defer a transaction with *a2* until *a1*'s transaction has finished.

Once a plan is built, it is executed by converting each action into its head, speech, and arm behaviours, based on a simple set of rules. Execution is monitored for plan correctness by comparing states from the state manager against states predicted by the planner. In the case of divergence, the planner is directed to construct a new plan using the sensed state as its new initial state. For example, if *a1*'s response to *ask-drink(a1)* was not understood, the execution monitor will direct PKS to build a new plan. One result is a modified plan that first informs *a1* they were not understood before repeating the *ask-drink* action and continuing the old plan.

Another consequence of execution monitoring is that certain types of overanswering can be detected and handled through replanning. For instance, a *greet(a1)* action by the robot might cause the customer to respond with an utterance that includes a drink order. In this case, the monitor would detect that the preconditions of *ask-drink(a1)* aren't met and direct PKS to replan. A new plan could then omit *ask-drink* and proceed to acknowledge and serve the requested drink.

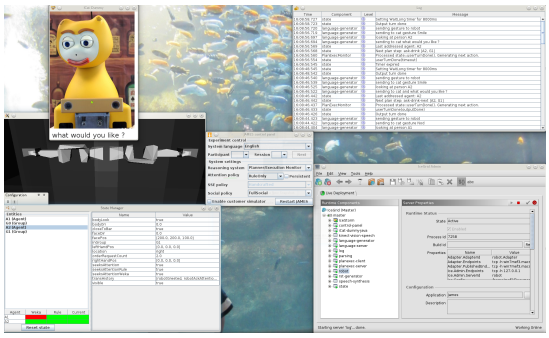

Figure 5: The JAMES software interface

The complete bartender system uses the physical or simulated robot to process interactions similar to those shown above. Users interact with the system using speech, while the main system interface (Figure 5) displays the reasoning and execution status of the core components, including planning and state management.

Acknowledgements

The authors thank their JAMES colleagues who helped implement the bartender system: Andre Gaschler, Manuel Giuliani, Amy Isard, Maria Pateraki, and Richard Tobin. This research has received funding from the European Union's 7th Framework Programme under grant number 270435.

References

Baltzakis, H.; Pateraki, M.; and Trahanias, P. 2012. Visual tracking of hands, faces and facial features of multiple persons. *Machine Vision and Applications* 23(6):1141–1157.

Foster, M. E. 2013. Evaluating engagement classifiers for a robot bartender. In submission.

Henning, M. 2004. A new approach to object-oriented middleware. *IEEE Internet Computing* 8(1):66–75.

Huth, K. 2011. Wie man ein Bier bestellt. MA thesis, Fakultät für Linguistik und Literaturwissenschaft, Universität Bielefeld.

Isard, A., and Matheson, C. 2012. Rhetorical structure for natural language generation in dialogue. In *Proceedings of SemDial 2012 (SeineDial)*, 161–162.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of AIPS 2002*, 212–221.

Petrick, R. P. A., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of ICAPS 2004*, 2–11.

Petrick, R. P. A., and Foster, M. E. 2013. Planning for social interaction in a robot bartender domain. In *Proceedings of ICAPS 2013, Special Track on Novel Applications*.

Rickert, M. 2011. *Efficient Motion Planning for Intuitive Task Execution in Modular Manipulation Systems*. Dissertation, Technische Universität München.

White, M. 2006. Efficient realization of coordinate structures in Combinatory Categorial Grammar. *Research on Language and Computation* 4(1):39–75.

Authoring Plan-based Narratives via a Social Network

Julie Porteous, Fred Charles and **Marc Cavazza**
School of Computing,
Teesside University,
Middlesbrough TS1 3BA,
United Kingdom
{j.porteous,f.charles,m.o.cavazza}@tees.ac.uk

Abstract

One way of interacting with an Interactive Storytelling system is via an authoring system prior to plan-based narrative generation. In the search for a user-friendly authoring method for plan-based storytelling domains we have developed a method that captures important narrative aspects such as characters' relationships as a way of defining a story. This represents a novel form of high-level authoring for plan-based storytelling which fits specific narrative genres: namely, serial dramas (or soap operas) where social relationships between characters act as a determinant for the narrative events that make up different episodes. The approach is implemented in a demonstration system which makes the dependency explicit: using a visual interface users can set social relationships between virtual characters and generate an episode based on that network. Stories are generated at run-time using a plan-based approach that exercises meta-level control over narrative trajectory via the use of pseudo-landmarks. Thus the system provides authors with a visual mechanism for the specification of key story determinants and observation of their impact on generated narratives. The demonstration system is set in the medical drama genre (in the style of serials such as *House, ER and Scrubs*). During the demo participants are able to interact freely with the system: setting relationships between virtual characters to "author" an episode of the drama in which the relationships they have set lead to peripeteia in the context of medical story lines; and then watching this episode as it is visualised as a 3D animation.

Introduction

In Interactive Storytelling systems, user interaction can occur at different stages: during the presentation of the story; and prior to story generation via an authoring system. Plan-based narrative generation has been shown to be applicable in both cases (Riedl and Young 2010; Porteous, Cavazza, and Charles 2010). In the search for a user-friendly authoring method that would capture important narrative aspects such as characters' relationships we have developed a system in which the characters' social network can be used to define a story. In this paper we present an interface for high-level authoring of plan-based stories that is targeted at those narrative genres, such as serial dramas and soap operas, where social relationships between characters act as determinants for the evolution of narrative across episodes.

This represents a novel mechanism for interactive narrative that reflects aspects of how modern dramas are shaped in specific genres, where situations and relationships are determinant. For example, advice in the contemporary film and screen writing literature advises authors to think initially, and perhaps primarily, of story in terms of characters, relationships and situations (McKee 1997; Phillips and Huntley 2009). This is the idea which we have explored in this work: to start from models of characters and the relationships between them, and then to explore the situations that can occur and the stories that will necessarily arise from that.

Our demonstration system is set in the medical drama genre where social relationships are in a constant process of dramatic change, where conflict dominates (Alexander et al. 1992; Greenberg, Abelman, and Neuendorf 1981) and which are known to elicit audience reactions to both dramatic events and character relationships (Bradley 2007). Interestingly, these genres are repetitive since they frequently feature different combinations of typical actions yet diversity is achieved via changes in the relationships between characters and the conflicts and situations that arise as a consequence of this (for example, series 1 of ER included repeated instances of: seduction, conflict over treatment, professional rivalry, battles to save patients and so on).

System Architecture Overview

The architecture of the system is represented in Figure 1 with the central components being the visual user interface (1), the plan-based narrative engine (2) and narrative visualizer (3), with co-ordination between them as shown.

User interaction with the system is via a graphical representation of a social network, representing the current state of the social relationships between virtual characters. This network has virtual characters as nodes (including their names and a picture of them), relationships between them as arcs and characters clustered according to their role such as junior doctors, patient relatives, nurses and so on. Due to the ubiquity of social networks the conceptual basis of this interaction mechanism is one that users are likely to be familiar with. Nevertheless it represents a novel form of interacting with a storytelling system and one which users should find compelling. For ease of use graph drawing and

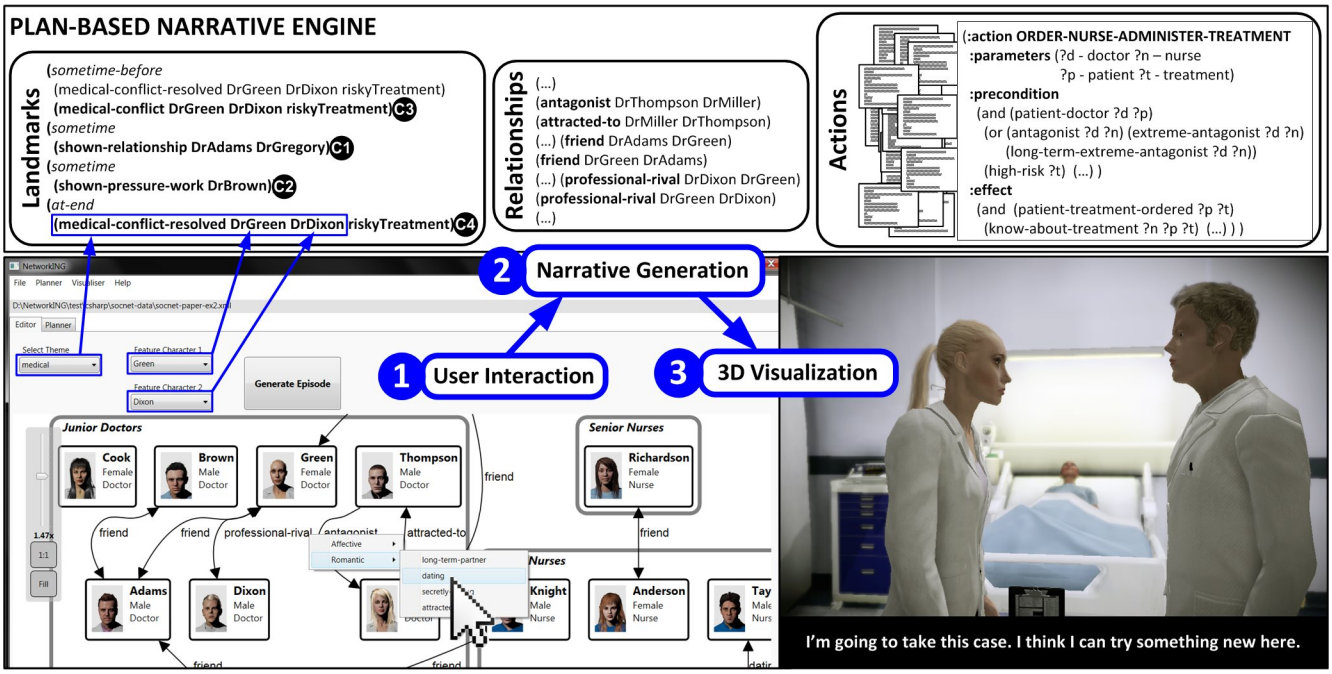

Figure 1: Architecture of the Demonstration System: social relationships specified via the interface (1) are used by the narrative engine in the creation of planning problem instances which also feature the use of constraints as landmarks for control during narrative generation (2). The output planned sequence of narrative actions are visualized on a 3D stage using Unreal® (3).

layout of the social network is handled automatically using (Graphviz4Net 2011) which can generate graphs where all elements are fully customizable.

When an episode of the medical drama is to be generated the current state of the social network is used in the creation of a planning problem instance. This problem instance then forms part of the input to the narrative generator at plan-time, along with the domain actions. The planning problem instance that is created includes PDDL3 modal operators which specify a partial order over dramatically interesting narrative situations and content. At run time these are used like landmarks to control the trajectory of the output plan, in a decomposition-based approach, as reported by Porteous et al (2010).

The generated episode of the medical drama is visualized by a component that receives narrative actions and stages them in a 3D environment using the Unreal® game engine (UDK). In the staging of the actions the notion of "Smithian" cues (Smith 2003) is employed to enhance important narrative events. These include such aspects as lighting, music, camera angles and shot distance.

Virtual character dialogues are generated by the system at run-time and are passed through a text-to-speech system that synchronizes spoken utterances with characters' lip-synching. The character dialogues are also displayed in the visualization window in the form of sub-titles.

Narrative Generation

Underpinning the system is a plan-based narrative generator featuring an implementation of Metric-FF (Hoffmann 2003) adapted to use landmarks for narrative control as described previously in (Porteous, Cavazza, and Charles 2010; Porteous et al. 2011). Their extension to planning with landmarks (Hoffmann, Porteous, and Sebastia 2004) provides a mechanism to ensure the inclusion of important dramatic points and their relative order within a narrative in a way that promotes story diversity whilst retaining the generative power of the approach. For the medical drama domain in which the demonstration system is set, such points of the drama can include tense clinical situations, strained relationships between characters, deceptions, confrontations and so on. Within this approach landmarks are represented

(sometime-before
(medical-conflict-resolved DrGreen DrDixon riskyTreatment)
(medical-conflict DrGreen DrDixon riskyTreatment))
(sometime
(shown-relationship DrAdams DrGregory))
(sometime
(shown-pressure-work DrBrown))
(at-end
(medical-conflict-resolved DrGreen DrDixon riskyTreatment))

Figure 2: Sample PDDL3 modelling of narrative landmarks

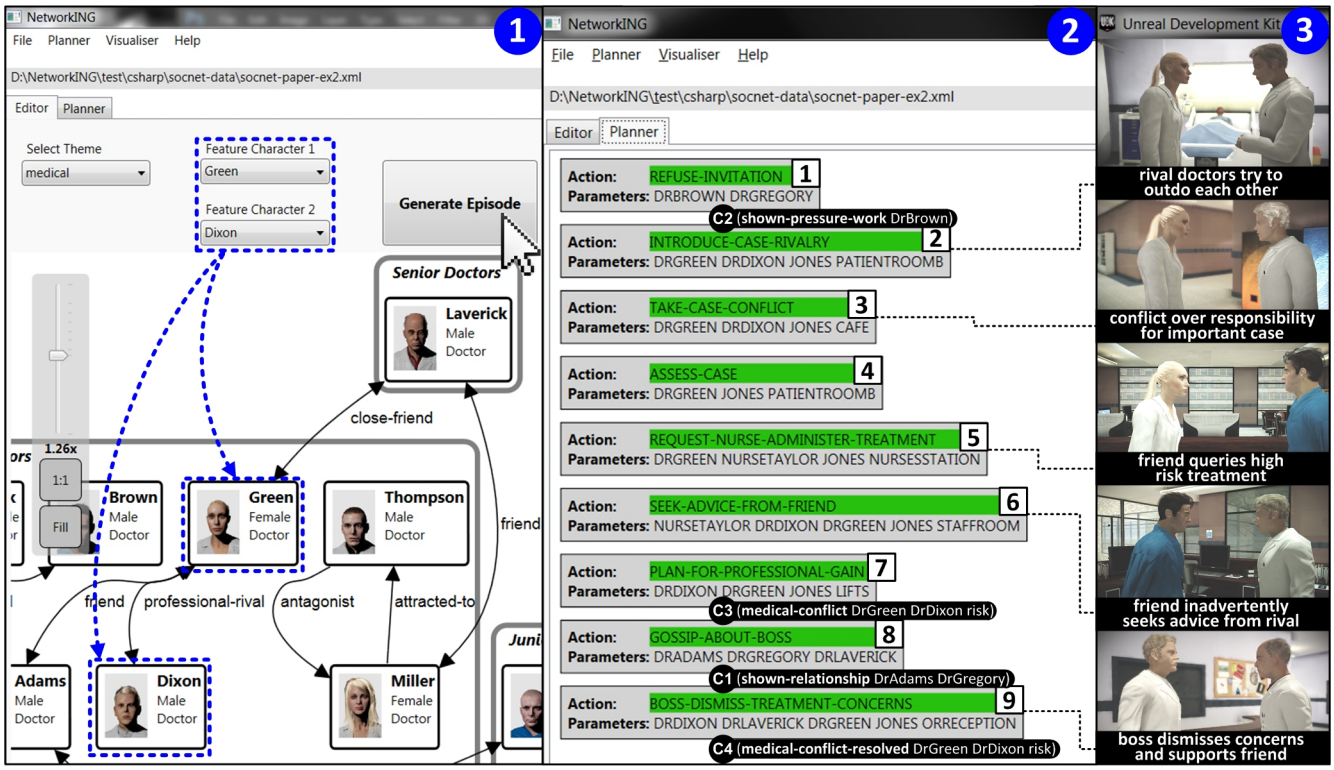

Figure 3: Overview of Interaction with the Demonstration System: (1) users specify relationships between characters, select feature characters and goal theme; (2) episode is generated using current state of the social network and constraints C1–C4 to structure narrative; (3) view a visualization of it as a 3D animation (illustrated with screenshots and brief plot synopses).

declaratively, with partial orders specified over them using PDDL3.0 modal operators such as *sometime-before*, *sometime* and *at-end* and then used in a decomposition based planning approach to control the shape of a narrative trajectory as it is generated. At run time these constraints are linearised and used to decompose the process of narrative generation into a sequence of sub-narratives. A complete output narrative is produced by conjunction of the sub-sequences.

As an example, the episode illustrated in Figure 3 is generated for a problem instance which includes the constrained landmarks shown in Figure 2. The use of these landmarks ensures the generation of a narrative that contains suitable dramatic content. A selection of actions illustrating this narrative episode are shown in Figure 3.

System Performance

System performance was analysed in (Porteous, Charles, and Cavazza 2013) through hundreds of system runs, in terms of real-time performance, story diversity and leverage effect of the modification of the social network onto the generated narratives. This approach preserves the run time performance of our baseline narrative engine while showing the potential for moderate changes to the social network to yield large changes across hundreds of narratives generated in our experiments.

Demonstration Overview

During an interactive session with the demonstration system users are given the opportunity to "author" their own episode of the medical drama by specifying the relationships between characters in the social network, and then watching it. Through this process they can explore the difference in narrative possibilities as a result of changes in relationships between characters.

Step 1: Specify Relationships between Characters

Via the interface users can add and delete virtual characters from the network, choosing from an available set that includes 10 doctors, 5 nurses, 3 patients and 3 relatives. Similarly users can add, delete and modify relationships between the characters choosing from a classification of affective and romantic relationships as detailed in (Porteous, Charles, and Cavazza 2013). For ease of use all interaction is mouse and menu driven, with the use of (Graphviz4Net 2011) which gives Windows WPF control over the interface.

Relationships between characters can be symmetric or asymmetric. For example, part 1 of Figure 3 includes a symmetric relationship between Dr Green and Dr Dixon (they are professional rivals) and an asymmetric relationship between Dr Thompson and Dr Miller (he is antagonistic to her whilst she is attracted to him).

Step 2: Generate the Episode

Once a user has specified relationships in the network they can then generate an episode of the medical drama. At this point they are required to select feature characters for their narrative episode and to choose a "goal theme" for their episode from a menu of possibilities such as romantic intrigue, medical issues, pressure of work and so on. Once these are specified they can select "Generate Episode" in the interface, as shown in Figure 3. They can also inspect the sequence of narrative actions that constitute the generated episode before watching its visualization (part (3) in Fig 3).

The relationships that the user has specified in the network impacts on the likelihood of different narrative events occurring so for example, if the user has set the relationship between a pair of characters to be antagonistic then the narrative is more likely to include confrontation between them, arguments, "ganging up" and so on. Inspection of the narrative at this point enables the user to assess its quality prior to watching the visualization. An example of a generated narrative is shown in part (2) of Figure 3.

Step 3: Watch the Episode

Once the narrative had been generated the user can select to view the episode and watch the visualization of it. As an illustration, part (3) of Figure 3 shows a series of screenshots from the visualization of the narrative generated for the configuration of the social network shown in part (1).

For this narrative the user selected Dr Green and Dr Dixon as the feature characters of interest and a medical theme for the goal of the narrative. It can be seen that the relationships specified by the user in the social network have a direct impact on the evolution and content of the narrative: the feature doctors are professional rivals and this is reflected in their interactions in the narrative which feature confrontation and a rival doctors plan to gain the upper hand when they spot the possibility. However in this instance the outcome of the narrative depends on relationships with other secondary characters: because the senior doctor, Dr. Laverick, brought in to the story is a close friend of Dr. Green they choose to support them rather than the rival doctor, Dr. Dixon.

Since the episode can also be viewed by other members of the audience the visualization will highlight key dramatic events so that the types of relationships between characters can be clearly recognized by the demo audience at large.

Conclusion

The approach implemented in our demonstration system represents a novel direction for narrative generation with a move towards a user-friendly authoring method that captures important narrative aspects – namely characters' relationships – in a way that reflects how modern dramas are shaped in genres where relationships are determinant.

Acknowledgments. This work was funded in part by the European Commission through the FP7 Open FET "MUSE" Project (ICT-296703). Visual content developed by Catherine Dixon and Matthew Laverick of Teesside University. Character models purchased from aXYZ Design http://www.axyz-design.com.

References

Alexander, A.; Carveth, R.; Bohrer, G.; and Ryan, M. 1992. Investigating Gender Difference in College Student Soap Opera Viewing. In *Staying Tuned: Contemporary Soap Opera Criticism*. Bowling Green SU Press.

Bradley, S. D. 2007. Examining the Eyeblink Startle Reflex as a Measure of Emotion and Motivation to Television Programming. *Communication Methods and Measures* 1:7–30.

Graphviz4Net. 2011. `http://graphviz4net.codeplex.com/`. [Last Accessed: 10-04-13].

Greenberg, B. S.; Abelman, R.; and Neuendorf, K. 1981. Sex on the Soap Operas: Afternoon Delight. *Journal of Communication* 31:83–89.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research (JAIR)* 22:215–278.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research* 20:291–341.

McKee, R. 1997. *Story: substance, structure, style, and the principles of screenwriting*. NY: ReganBooks.

Phillips, M., and Huntley, C. 2009. *Dramatica – A New Theory on Story*. Write Brothers Press; 10th Anniversary Edition edition.

Porteous, J.; Teutenberg, J.; Pizzi, D.; and Cavazza, M. 2011. Visual Programming of Plan Dynamics using Constraints and Landmarks. In *Proc. of the 21st Int. Conf. on Automated Planning and Scheduling (ICAPS 2011)*, 186–193.

Porteous, J.; Cavazza, M.; and Charles, F. 2010. Applying Planning to Interactive Storytelling: Narrative Control using State Constraints. *ACM Transactions on Intelligent Systems and Technology (ACM TIST)* 1(2):1–21.

Porteous, J.; Charles, F.; and Cavazza, M. 2013. NetworkING: using Character Relationships for Interactive Narrative Generation. In *Proc. of 12th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS 2013)*, 595–602. IFAAMAS.

Riedl, M. O., and Young, R. M. 2010. Narrative Planning: Balancing Plot and Character. *Journal of Artificial Intelligence Research* 39:217–267.

Smith, G. 2003. *Film Structure and the Emotion System*. Cambridge University Press.

A generic constraint-based local search library for the management of an electromagnetic surveillance space mission

Cédric Pralet and **Guillaume Infantes** and **Gérard Verfaillie**
ONERA - The French Aerospace Lab, F-31055, Toulouse, France
{Cedric.Pralet,Guillaume.Infantes,Gerard.Verfaillie}@onera.fr

Abstract

This paper presents what has been done at the French Aerospace Lab (ONERA) to deal with a scenario of space mission defined by the French Space Agency (CNES). This space mission is dedicated to the surveillance from space of ground electromagnetic sources. It involves two satellites: one for source detection and another one for data acquisition and download. It presents two sources of uncertainty: the presence or not of electromagnetic sources and, in case of presence, the volume of data generated by acquisition. Due to these uncertainties and to limited communication windows with ground control stations, online planning and scheduling (P&S) is necessary on board the second satellite to make consistent and optimal decisions in terms of data acquisition and download. In this paper we show how a generic constraint-based local search library can be used to build the onboard planning and scheduling component. This library, called *InCELL*, has been developed at ONERA. It allows temporal constraints, resource constraints, arithmetic and logical constraints, and optimization criterion to be quickly and incrementally evaluated at each step of a local search algorithm. Already experimented to deal with simpler scenarios, this is the first time it is experimented on a complex scenario involving agile satellites. We show also how the generic simulation tool *Ptolemy* can be used to simulate the space system and evaluate its P&S component.

Introduction

In the context of the CNES-ONERA *Agata* project about spacecraft autonomy (Charmeau and Bensana 2005), after working on a first mission scenario involving only one non agile Earth optical detection and observation satellite (Damiani, Verfaillie, and Charmeau 2004; Pralet and Verfaillie 2008), ONERA dealt with a more complex mission scenario defined by CNES and called *Agata-One*. The main objective was to assess whether or not the tools that were defined to deal with the first scenario can be easily adapted to deal with a more complex one.

The *Agata-One* scenario involves two agile Earth satellites placed on low altitude, circular orbits, on the same orbital plane. Agile satellites are able to perform very quick attitude movements along the three axes around their gravity center (roll, pitch, and yaw) generally thanks to gyroscopic actuators which are more efficient than usual reaction wheels. Thanks to regular roll attitude movements, the first satellite (S_1) scans a wide strip around its ground track. Thanks to its instruments, it is able to detect the presence of electromagnetic sources at the Earth surface and to localize them. In case of detection, it sends instantaneously information to the second satellite (S_2, which follows it at a small distance) via a permanent inter-satellite low-rate communication link. Satellite S_2 maintains a set of ground areas on which electromagnetic sources have been detected. Each time it overflies one of these areas, it can acquire data from it. To do that, it must perform a roll and pitch attitude movement to direct its acquisition instrument (a reception antenna) towards this area (the reception antenna is body-mounted on the satellite). When too many close areas must be handled, it must decide on those it will effectively handle and on the acquisition order. Once data from an area has been acquired, it is memorized in a mass memory and downloaded to ground reception and processing stations via a non permanent satellite-ground high-rate communication link. Downloading data to a ground station is only possible within one of the station visibility windows. Moreover, it is only possible when the satellite attitude is compatible with data download (as the reception antenna, the emission antenna is body-mounted on the satellite and, during the whole download period, the station must remain inside the satellite emission antenna cone). As for data acquisition, when too much data must be downloaded, satellite S_2 must decide on those it will download and on the download order. Fig. 1 summarizes the exchanges between satellites and ground reception stations.

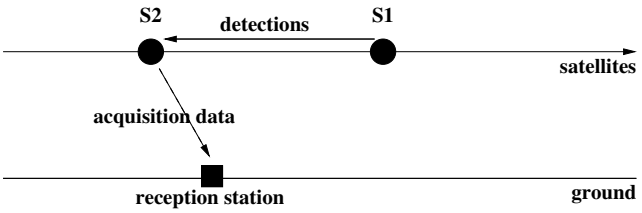

Figure 1: Exchanges between satellites and ground reception stations.

It must be stressed that the attitude of satellite S_2 allow-

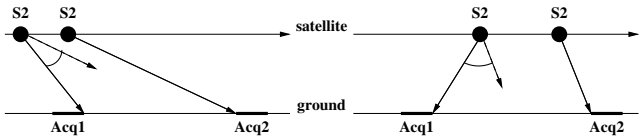

Figure 2: How the attitude movement to be performed by satellite S_2 to transit from a data acquisition to another one depends on the time at which the transition is triggered. In the second case (right), the angular movement to be performed is greater than in the first case (left).

ing it to direct its reception antenna towards a given area depends on the position of the satellite on its orbit and thus on time. In such conditions, the attitude movement necessary to transit from a data acquisition from a given area to a data acquisition from another area, and thus the time taken by this transition, depends not only on both areas, but also on the time at which the transition is triggered (time-dependent transition duration). See Fig. 2 for an illustration.

This mission scenario involves two main sources of uncertainty: the presence or not of electromagnetic sources and, in case of presence, the volume of data generated by acquisition (this volume is highly variable and can typically range from 1 to 1000). Due to these uncertainties and to the non permanent visibility of satellites by ground control stations, online decision-making on data acquisition and download is necessary on board satellite S_2. To make such decisions, it would be possible to use manually defined decision rules. However, decisions would be better informed if they could use the result of P&S: planning and scheduling regularly performed over a given horizon ahead, using the most up to date information about detections and data volumes; decisions made according to the first steps of the plans produced.

Due to the limited computing time available for P&S and due to the limited computing resources available on board (CPU and RAM), heuristic search (greedy and/or local search) seems to be the right option to build an anytime combinatorial search procedure, able to produce quickly good quality plans and to improve on them as long as time is available before making decisions. It is widely used in space missions that require online onboard P&S (Chien et al. 2000; 2005b; 2005a). We already used it in the context of Earth observation and surveillance missions (Lemaître et al. 2002; Beaumet, Verfaillie, and Charmeau 2011; Pralet and Verfaillie 2008; Pralet et al. 2011; Verfaillie et al. 2011). However, each time, we built specific heuristic search procedures, dedicated to the specific mission at hand and not directly reusable to handle other missions. To deal with the *Agata-One* scenario, we decided to change our approach and to use generic tools developed at ONERA in the context of the *Agata* project and, more specifically, the *Invariant-based Constraint EvaLuation Library* (*InCELL* (Pralet and Verfaillie 2013)).

InCELL draws its inspiration from the ideas of *Constraint-based local search* (CLS (Hentenryck and Michel 2005)). In CLS, the user defines a model of its problem in terms of decision variables, constraints, and optimization criterion. She/he defines also its local search procedure over the set of complete variable assignments (where every variable is assigned). Because the speed of each local move is one of the keys to local search success, the software uses so-called *invariants* which allow expressions and constraints to be quickly and incrementally evaluated after each move. In *InCELL*, multiple-input multiple-output invariants allow expressions, arithmetic and logical constraints, temporal and resource constraints to be expressed and efficiently handled. *InCELL* calls for *Simple Temporal Network* (STN (Dechter, Meiry, and Pearl 1991)) techniques which allow temporally flexible plans to be produced, and for *Time-dependent* STN (TSTN (Pralet and Verfailllie 2012)) techniques which allow time-dependent transition durations to be taken into account.

To deal with the *Agata-One* scenario, an *InCELL* model of the associated P&S problem (decisions about data acquisition and download by satellite S_2) was built, a simple non chronological greedy search procedure was designed, and the events that trigger a new call to P&S over a given horizon ahead were defined.

To simulate the space system and to evaluate its P&S component, an event-based model of the system, based on the notions of state, event preconditions and effects, and event activations, was built and implemented using the generic simulation tool *Ptolemy* (Eker et al. 2003). Whereas P&S allows only the utility of decisions over the planning horizon to be evaluated, this simulation allows the global utility of successive decisions over the simulation horizon to be evaluated.

Sect. 1 describes problem data and Sect. 2 presents the structure of possible decisions. In Sect. 3, a constraint-based model of the P&S problem is introduced. The main ingredients of the *InCELL* library, as well as its main reasoning mechanisms, are presented in Sect. 4. Sect. 5 describes the search procedure and Sect. 6 defines when P&S is called. Sect. 7 shows how the space system and its P&S component can be simulated and evaluated, using the *Ptolemy* tool.

1 Problem data

Permanent (static) problem data is the following:

- a finite set of ground areas that must be kept under surveillance;
- a finite sequence of priority levels;
- for each ground area, its priority level, its weight (to give more or less weight to areas of the same priority level), an acquisition duration, an expected, a minimum, and a maximum volume of data resulting from acquisition;
- a finite set of ground reception stations;
- a data download rate from satellite S_2 to any ground reception station.

Moreover, it is assumed that a function associates with each ground area a and each time t the attitude of satellite S_2 necessary to acquire data from a at time t, when acquisition is possible, and that another function associates with each pair of attitudes of satellite S_2 the minimum time necessary to reach the second one, starting from the first one.

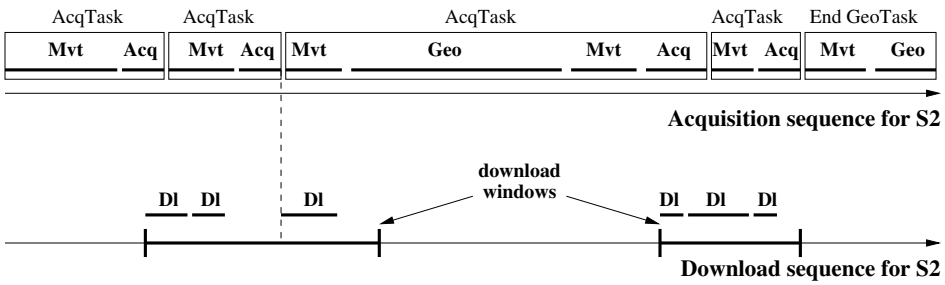

Figure 3: The two concurrent sequences of action on board satellite S_2. AcqTask = acquisition task; Geo Task = geocentric task; Mvt = attitude movement; Acq = data acquisition; Geo = geocentric pointing; Dl = data download.

Each time P&S is called, its complementary (dynamic) data is the following:

- a planning horizon ahead;
- an attitude of satellite S_2 at the beginning of the planning horizon;
- a set of ground areas from which electromagnetic sources have been detected by satellite S_1, but no acquisition by satellite S_2 has been performed yet;
- for each of these ground areas, its detection time and a finite sequence of acquisition windows by satellite S_2 over the planning horizon;
- a finite set of acquisitions that have been already performed, but whose data has not been downloaded yet (still present in memory);
- for each of these acquisitions, its detection and acquisition times and its actual volume in memory;
- a finite sequence of download windows by satellite S_2 over the planning horizon.

Acquisition windows are reduced in case of intersection with a download window, when acquisition is incompatible with download, in order to give priority to data download.

2 Possible decisions

On board satellite S_2, it is necessary to decide on two concurrent sequences of action:

- the sequence of data acquisitions;
- the sequence of data downloads.

The first sequence is made of acquisition tasks, each one following immediately the previous one. An acquisition task is, according to an HTN-like decomposition of tasks into sub-tasks (*Hierarchical Task Networks* (Nau et al. 2003)), itself made of:

- either an attitude movement immediately followed by a data acquisition;
- or an attitude movement immediately followed by a geocentric pointing, immediately followed by another attitude movement, immediately followed by a data acquisition (satellite geocentric pointing maintained towards Earth center is a waiting action, favourable to communication with Earth, data downloads, and energy recharging).

This sequence, possibly completed by an attitude movement followed by a geocentric pointing at the end of the planning horizon, entirely defines the attitude trajectory of satellite S_2.

The second sequence is made of data downloads, each one being performed within a download window. In this sequence, a download may not immediately follow the previous one. This is the case when it is necessary to wait for the end of an acquisition before downloading resulting data within a download window.

Fig. 3 illustrates the two concurrent sequences of action. Both sequences are not independent from each other because data download requires preceding acquisition.

3 A constraint-based model

P&S problem is a kind of over-constrained scheduling problem (over-constrained because it may be impossible to schedule all the candidate tasks (Kramer and Smith 2003)) which can be modeled using only constraints over intervals. An interval is defined by its presence, its starting date, its ending date, and its duration. Its presence is a boolean, equal to 1 if and only if the interval is effectively present in the schedule.

The model associates:

- with each ground area from which electromagnetic sources have been detected by satellite S_1, but no acquisition by satellite S_2 has been performed yet, an acquisition interval, a geocentric pointing interval, and a download interval;
- with each acquisition that has been already performed by satellite S_2, but whose data has not been downloaded yet, a download interval.

These intervals may be present or absent. Constraints to be satisfied are the following:

- each acquisition interval must be, when present, included in one of the acquisition windows of the associated ground area; its duration is the acquisition duration of the associated ground area, defined in the problem data;
- each download interval must be, when present, included in one of the download windows; if the acquisition has been already performed at the P&S time, download duration is equal to the actual volume in memory divided by

the download rate; if it has not been performed yet, it is equal to the maximum volume resulting from acquisition divided by the download rate (pessimistic assumption allowing the produced schedule to be surely executed);

- for each ground area from which electromagnetic sources have been detected, absence of the acquisition interval implies absence of the geocentric pointing and download intervals; presence of the geocentric pointing interval implies that it must precede the acquisition interval and follow the previous acquisition interval; presence of the download interval implies that it must follow the acquisition interval;
- there must be no overlapping between present acquisition and geocentric pointing intervals and enough time between successive intervals to allow attitude movements; moreover movements to or from geocentric pointings must be performed in minimum time in order to give geocentric pointing as much time as possible;
- there must be no overlapping between present download intervals.

The criterion to be optimized is a vector of global utilities, one per priority level. The global utility associated with a priority level p is equal to the sum of the local utilities associated with each of the ground areas of priority p. The local utility associated with a ground area is equal to its weight multiplied by two functions which both take a value between 0 and 1: a decreasing function of the time between detection and acquisition and another decreasing function of the time between acquisition and download. These functions tend to encourage quick acquisition and quick delivery of information on the ground. Two vectors of global utilities, resulting from two schedules, are lexicographically compared from the highest priority level to the lowest one.

4 The InCELL library

InCELL (*Invariant-based Constraint EvaLuation Library*) is a software library, dedicated to the quick incremental evaluation of expressions and constraints.

InCELL draws its inspiration from the ideas of *Constraint-based local search* (CLS (Hentenryck and Michel 2005)). In CLS, the user defines a model of its problem in terms of decision variables, constraints, and optimization criterion. She/he defines also its local search procedure over the set of complete variable assignments (every variable assigned). Because the speed of each local move is one of the keys to local search success, the software uses so-called *invariants* which allow expressions and constraints to be quickly and incrementally evaluated after each move. An invariant is a one-way constraint of the form $x \leftarrow exp$, where x is a variable and exp a function of other variables, such as for example $x \leftarrow \sum_{i=1}^{N} y_i$. On this example, when the value of y_j for some j is modified, it is not necessary to recompute $\sum_{i=1}^{N} y_i$ from scratch. It suffices to add to the previous value of x the new value of y_j, minus its old value. The only condition is the absence of cycles in the definition of invariants (no variable directly or indirectly function of itself).

InCELL extends the definition of invariants by allowing multiple-input multiple-output invariants. Invariants allow expressions, but also constraints, to be represented. Constraints, such as for example $\sum_{i=1}^{N} y_i \leq K$, are specific invariants whose evaluation stops when they are violated. In *InCELL*, a constraint optimization problem (variables, constraints, and criterion) takes the form of a DAG (*Directed Acyclic Graph*) of invariants. Each time the value of some atomic variables (variables that are not functions of other variables and are roots of the DAG) is modified, the DAG of invariants is lazily reevaluated according to a DAG topological order: any invariant is reevaluated only when necessary and at most once.

On top of these basic concepts and mechanisms, *InCELL* offers some constructs dedicated to scheduling: time point variables, interval variables (defined by two time point variables and a distance constraint between them), unary and binary distance constraints (of the form $x \leq K$ or $x - y \leq K$). All temporal constraints are managed using a special STN invariant (*Simple Temporal network* (Dechter, Meiry, and Pearl 1991)) which has as inputs a set of unary and binary distance constraints and as outputs the earliest dates of all the time point variables involved in the constraints. Classical STN techniques are used to handle the STN: constraint propagation, maintenance of propagation chains, decomposition of the distance graph into strongly connected components. Moreover, STN concepts and techniques are extended in *InCELL* to deal with so-called *time-dependent scheduling* (Gawiejnowicz 2008), that is with time-dependent distance constraints (Pralet and Verfailllie 2012) where the minimum distance is not a constant, but a function of the involved time points (of the form $x - y \leq F(x, y)$ with some assumptions about Function F). All the constraints over intervals, defined in the previous section, can be managed by *InCELL*, including the minimum transition times between successive acquisition and geocentric pointing intervals, thanks to time-dependent distance constraints.

InCELL allows also resource constraints to be defined and profiles of resources (with piecewise constant or linear evolutions) to be quickly and incrementally maintained, taking into account the earliest dates produced by the STN. This would allow memory (piecewise constant evolution) and energy (piecewise linear evolution) constraints to be managed. However, these constraints are ignored in our problem: energy because it is not limiting and memory because of the uncertainty about the volume of data generated by acquisition. When planning acquisitions, we prefer not to limit acquisitions because of possible large volumes of data. However, when executing the acquisition plan, before triggering an acquisition, in case of possible memory overflow, we remove from memory lower priority data and, when it is not sufficient, we cancel the acquisition.

One of the key features of *InCELL* is its ability to work on dynamic models (a new model each time P&S is called) using a unique static model which is recycled to build dynamic models. This allows any dynamic memory allocation to be avoided on board: a key requirement when building embedded reactive control software.

See (Pralet and Verfaillie 2013) for more details about the

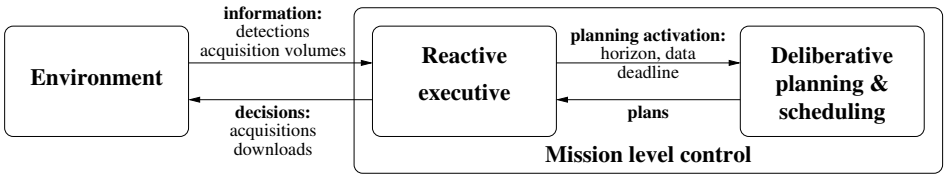

Figure 4: Exchanges between the environment, the reactive executive, and the deliberative P&S component.

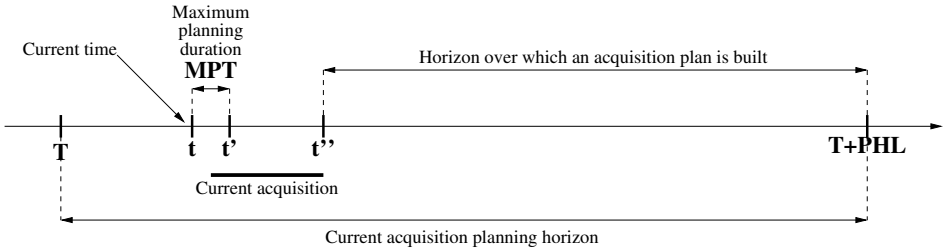

Figure 5: Horizon over which an acquisition plan is built at a given time t on board satellite S_2.

InCELL library.

5 A non chronological greedy search

To build online acquisition and download plans on board satellite S_2, we defined a very simple greedy search procedure, although more sophisticated local search procedures could be considered (Aarts and Lenstra 1997).

At each step of this procedure, an acquisition (resp. download) of highest priority level and of highest utility at this priority level is selected and added to the acquisition (resp. download) plan, when addition is possible. It is added in the best acquisition (resp. download) window and at the best position in the acquisition (resp. download) sequence in terms of utility. Once the acquisition sequence is defined, geocentric pointings are added between acquisitions, when possible.

6 Calls to planning and scheduling

In case of online P&S, it is not only necessary to define the P&S model and the reasoning and search mechanisms. It is necessary to define when the executive calls to P&S, in order to get a plan over some planning horizon ahead and to follow it until a new call to P&S. See Fig. 4 for a global view of the exchanges between the environment, the reactive executive, and the deliberative P&S component.

In our problem, as far as acquisitions are concerned, we define the length PHL of the planning horizon (horizon over which P&S is called; typically some hours) and the maximum planning time MPT (maximum time taken by P&S; typically some seconds). The planning horizon of length PHL is regularly shifted (typically every half an hour). P&S is called again when a new acquisition opportunity appears over the planning horizon. This happens either when electromagnetic sources are detected by satellite S_1 on some ground area, or when the planning horizon is shifted and a new acquisition window for some ground area appears over the new planning horizon. In such a case, we consider the current time t, the time $t' = t + MPT$ at which a plan will be surely available, the time t'' from which decisions can be made, taking into account acquisition or attitude movement possibly in progress at t' (acquisitions and attitude movements are not interruptible, but geocentric pointings are), and we call to P&S over the planning horizon from t''. See Fig. 5 for an illustration.

As far as downloads are concerned, P&S is called MPT before each download window (or group of windows that overlap or are very close to each other) over the whole window (or group of windows).

7 Simulation

We used the simulation tool *Ptolemy* to simulate the space system. *Ptolemy* (Eker et al. 2003) (see http://ptolemy.eecs.berkeley.edu/) is a generic tool dedicated to the simulation of dynamic systems, with an emphasis on hybrid simulation. Among many other possibilities, it is possible within *Ptolemy* to simulate a system whose dynamics involves both discrete events and continuous evolutions of resources. To express in *Ptolemy* the dynamics of the space system, we particularly relied on the *Ptera* framework (Feng, Lee, and Schruben 2010) which is based on the notions of state, events, event preconditions and effects, and conditional activations by events of other events (possibly with some delay and some probability). See Fig. 6 for an illustration of the several temporal horizons that are handled in the simulation (simulation, commitment, planning, and decision horizons).

This simulation was run on scenarios built by CNES. Fig. 7 shows a screenshot of the simulation tool at the end of a five day simulation horizon, where one can see:

- at the top left, the current acquisition requests over the whole world (small circles) and the visibility circle of the unique ground reception station (in blue);
- at the top right, an artist view of satellites S_2 (in front) and S_1 (behind) with the pointing direction of the former;

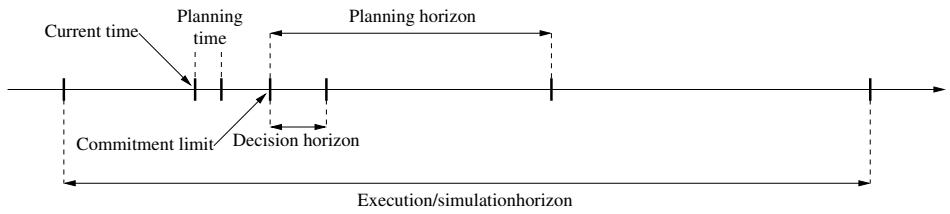

Figure 6: Illustration of the several temporal horizons that are handled in the simulation.

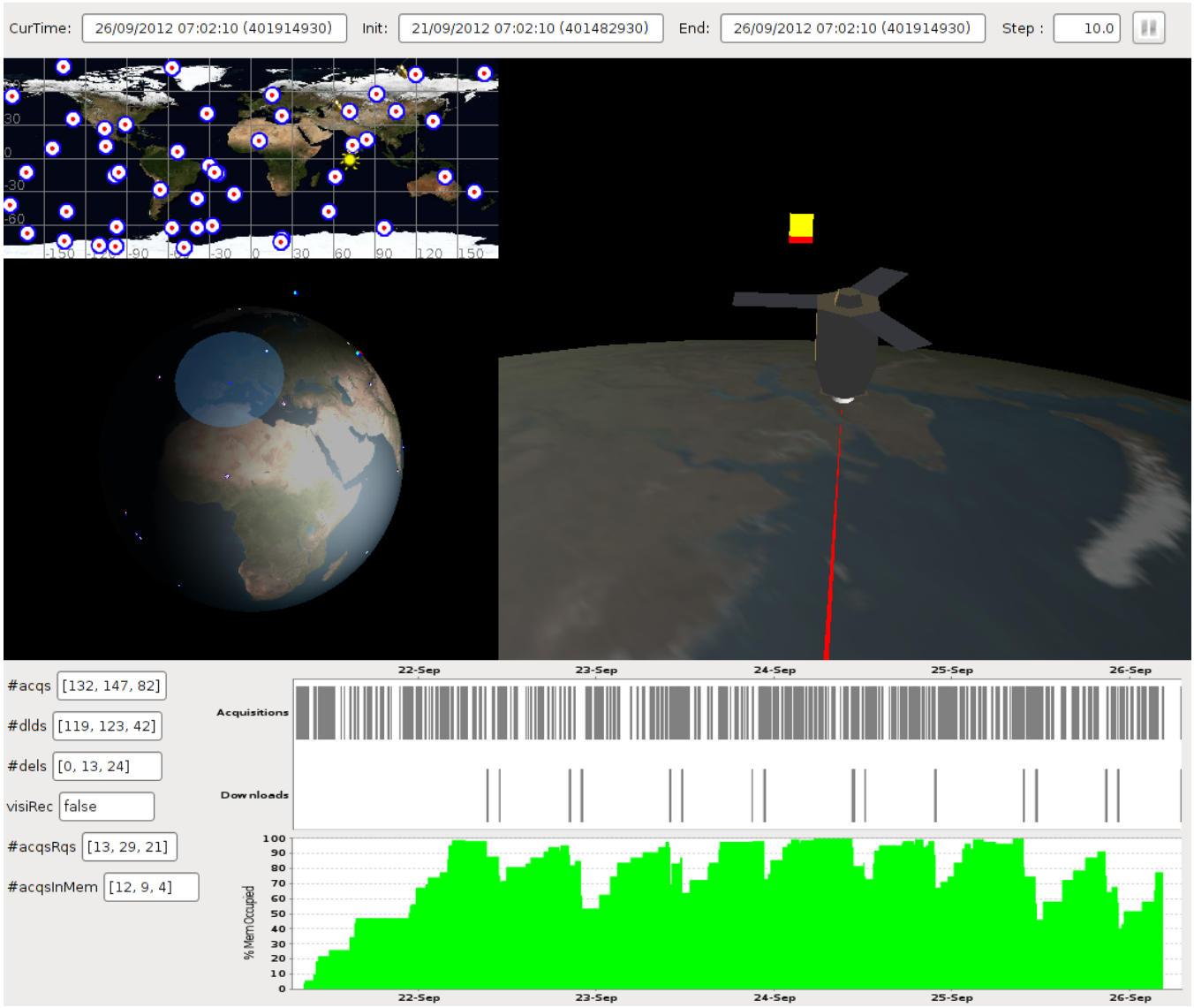

Figure 7: Screenshot of the *Agata-One* simulator at the end of a five day simulation horizon.

Prio	1	2	3
NAcq	132	147	82
NDl	119	123	42
NRm	0	13	24

Table 1: Global results over the five day simulation horizon: **Prio** = priority level, **NAcq** = number of performed acquisitions; **NDl** = number of downloaded acquisitions, **NRm** = number of performed acquisitions that have been removed from memory to free space.

- at the bottom right, the sequence of acquisitions, the sequence of downloads, and the evolution of memory on board.

Over this simulation horizon, acquisition planning is called 680 times, each time over a four hour horizon ahead. Download planning is called 16 times, each time over the next download window. Each time it is called, acquisition (resp. download) planning must manage some tens of acquisitions to be performed (resp. downloaded). Acquisition (resp. download) planning takes on average 432 ms (resp. 2258 ms) on an i5-520 Intel processor with 1.2 GHz and 4 GBRAM.

The global results per priority level are shown on Tab. 1 The mean utilization percentage of the downloads windows is of 85.39%.

A demonstration of the space system simulation is presented in the ICAPS 2013 Application Showcase.

Conclusion

The first result of this study is the demonstration that the generic *InCell* library allows the planning problem associated with a new complex space mission to be easily modeled and efficiently solved. Beyond the necessary improvements of the library in terms of modeling power and algorithm efficiency, the next steps should be the management of other missions, the implementation of the executive and of the P&S component on actual space processors, and the effective use on board an autonomous spacecraft, for example to manage data downloads in presence of uncertainty about volumes.

References

Aarts, E., and Lenstra, J., eds. 1997. *Local Search in Combinatorial Optimization*. John Wiley & Sons.

Beaumet, G.; Verfaillie, G.; and Charmeau, M. 2011. Feasibility of Autonomous Decision Making on board an Agile Earth-observing Satellite. *Computational Intelligence* 27(1):123–139.

Charmeau, M.-C., and Bensana, E. 2005. AGATA: A Lab Bench Project for Spacecraft Autonomy. In *Proc. of the 8th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-05)*.

Chien, S.; Knight, R.; Stechert, A.; R.Sherwood; and Rabideau, G. 2000. Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. In *Proc. of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*, 300–307.

Chien, S.; Cichy, B.; Davies, A.; Tran, D.; Rabideau, G.; Castano, R.; Sherwood, R.; Mandl, D.; Frye, S.; Shulman, S.; Jones, J.; and Grosvenor, S. 2005a. An Autonomous Earth-Observing Sensorweb. *IEEE Intelligent Systems* 20(3):16–24.

Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Mandl, D.; Frye, S.; Trout, B.; Shulman, S.; and Boyer, D. 2005b. Using Autonomy Flight Software to Improve Science Return on Earth Observing One. *Journal of Aerospace Computing, Information, and Communication* 2:196–216.

Damiani, S.; Verfaillie, G.; and Charmeau, M.-C. 2004. An Anytime Planning Approach for the Management of an Earth Watching Satellite. In *Proc. of the 4th International Workshop on Planning and Scheduling for Space (IWPSS-04)*.

Dechter, R.; Meiry, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49:61–95.

Eker, J.; Janneck, J.; Lee, E.; Liu, J.; Liu, X.; Ludvig, J.; Neuendorffer, S.; Sachs, S.; and Xiong, Y. 2003. Taming Heterogeneity: the Ptolemy Approach. *Proceedings of the IEEE* 91(1):127–144.

Feng, T.; Lee, E.; and Schruben, L. 2010. Ptera: An Event-oriented Model of Computation for Heterogeneous Systems. In *Proc. of the 10th International Conference on Embedded Software (EMSOFT-10)*, 219–228.

Gawiejnowicz, S. 2008. *Time-dependent Scheduling*. Springer.

Hentenryck, P. V., and Michel, L. 2005. *Constraint-based Local Search*. MIT Press.

Kramer, L., and Smith, S. 2003. Maximizing Flexibility: A Retraction Heuristic for Oversubscribed Scheduling Problems. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 1218–1223.

Lemaître, M.; Verfaillie, G.; Jouhaud, F.; Lachiver, J.-M.; and Bataille, N. 2002. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology* 6:367–381.

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Pralet, C., and Verfaillie, G. 2008. Decision upon Observations and Data Downloads by an Autonomous Earth Surveillance Satellite. In *Proc. of the 9th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-08)*.

Pralet, C., and Verfaillie, G. 2013. Dynamic Online Planning and Scheduling using a Static Invariant-based Evaluation Model. In *Proc. of the 23rd International Conference on Automated Planning and Scheduling (ICAPS-13)*.

Pralet, C., and Verfaillllie, G. 2012. Time-Dependent Simple Temporal Networks. In *Proc. of the 18th International Conference on Principles and Practice of Constraint Programming (CP-12)*, 322–338.

Pralet, C.; Verfaillie, G.; Olive, X.; Rainjonneau, S.; and Sebbag, I. 2011. Planning for an Ocean Global Surveillance Mission. In *Proc. of the 7th International Workshop on Planning and Scheduling for Space (IWPSS-11)*.

Verfaillie, G.; Infantes, G.; Lemaître, M.; Théret, N.; and Natolot, T. 2011. On-board Decision-making on Data Downloads. In *Proc. of the 7th International Workshop on Planning and Scheduling for Space (IWPSS-11)*.

Integrated Operations (Re-)Scheduling from Mine to Ship

S Kameshwara, Alfiya Tezabwala
IBM Research - India
Bangalore, India

Alain Chabrier
IBM Software Group
Spain

Julain Payne, Fabio Tiozzo
IBM Software Group
France

Abstract[1]

Mining companies have complex supply chains that start from the mining location and stretch thousands of kilometers to the end customer in a different country and continent. The logistics of moving the materials from mines to ship is composed of series of optimization problems like berth allocation, ship scheduling, stockyard scheduling, and rail scheduling, which are individually NP-hard. In this paper, we present a scheduling application, called as IBM Optimization: Mine to Ship, for end-to-end integrated operations scheduling. The application is built on IBM ILOG ODM Enterprise with advanced features like rescheduling under deviations and disturbances, and maintenance scheduling. The modeling and computational complexity of integrated scheduling optimization is tamed using hybrid optimization technique that leverages mathematical programming and constraint programming. The application will benefit the mining companies with increased resource usage, higher throughput, reduced cost of operations, and higher revenue.

[1]The paper is published in the Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013).

Coordinating Maintenance Planning under Uncertainty

Joris Scharpff and **Matthijs T.J. Spaan** and **Leentje Volker** and **Mathijs de Weerdt**
{j.c.d.scharpff, m.t.j.spaan, l.volker, m.m.deweerdt}@tudelft.nl
Delft University of Technology, The Netherlands

Abstract

In maintenance of (public) infrastructures, such as the national highway network, an asset manager is responsible for high network quality and throughput, while limiting expenses to a minimum. The maintenance activities, however, are often performed by commercial contractors, mainly driven by profit. Using a network-based payment mechanism we align the objectives of both stakeholders. Nonetheless, this greatly increases the complexity of planning maintenance activities, rendering it very difficult for human planners to develop (near-)optimal maintenance plans.

We demonstrate a support tool that facilitates multiagent planning for contractors so that they can coordinate their activities with other contractors in the network. This tool is initially intended as a serious game to create awareness and support amongst practitioners concerning this novel network-based coordination. In later stages we foresee great potential in the use of our tool as part of future dynamic contracting procedures.

Introduction

The planning and scheduling of maintenance activities on infrastructural networks, such as the highway network example of Figure 1 used in our gaming sessions, is a challenging real-world problem. While improving the quality of the infrastructure, maintenance causes temporary capacity reductions throughout the network. Given the huge impact of time lost in traffic on the economic output of a society, planning maintenance activities in a way that minimises the disruption of traffic flows poses an important challenge.

A powerful real-world example is the Summer 2012 closure of the A40 highway in Essen, Germany (Der Spiegel 2012). Instead of restricting traffic to fewer lanes for 2 years (the usual approach), authorities fully closed a road segment for 3 months, diverting traffic to parallel highways. Traffic conditions on the other highways hardly worsened, while €3.5M in social costs due to traffic jams were avoided (besides lowering construction costs).

Such convincing examples have motivated research into more innovative contracting procedures for infrastructural maintenance. In previous work (Volker et al. 2012), we presented a two-phase, dynamic contracting procedure as a solution for these problems. In the first phase, known as the procurement phase, maintenance activities are identified and assigned to the contractors through tendering. These activities are planned and performed in the second phase such that the inconvenience for the users of the network is limited to a minimum. The work we present here is a first step towards integration of our tool in the execution phase of this dynamic contracting procedure.

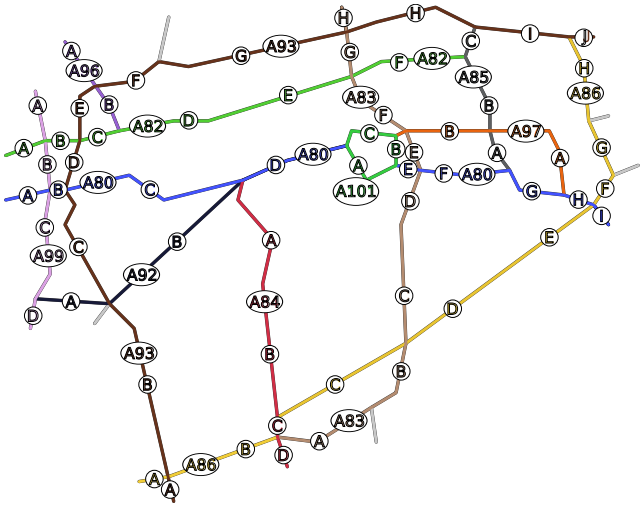

Figure 1: Example of a road network we use in the demonstration.

Problem Domain

We identified the need for a network-based approach towards maintenance planning. However, there are several complicating factors in this domain.

Firstly, while a (public) asset manager is commonly responsible for the quality and throughput of the network, the actual maintenance has to be performed by commercial and autonomous third-party contractors, mainly focused on maximising profits. These two objectives have to be aligned through rewards/penalties in order to steer towards socially favourable maintenance plans.

Secondly, contractors performing the maintenance are interdependent through their activities on the network. A contractor servicing one part of the network influences other contractors in other parts, as his work has a negative im-

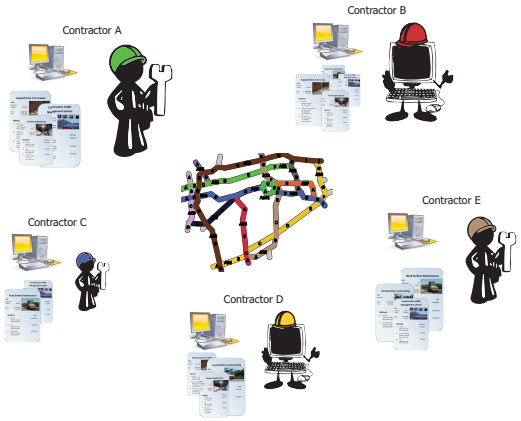

Figure 2: In the game, contractors (played by humans or computer agents) need to plan their given portfolio of activities on the network in the most profitable way. Their portfolios are represented by task cards, that specify the details of each activity. Note that each contractor is responsible for a different part of the network detailed in Figure 1.

pact on the traffic flow. Therefore, the use of congestion payments might result in high penalties for all contractors and hence the need for contractors to coordinate their maintenance plans on a network level is apparent.

Finally, execution of maintenance is inherently contingent. Apart from the possible difficulty of ascertaining an asset's actual maintenance state there are various causes for possible delays (e.g., weather, breakdowns, etc.).

Recently, we have proposed a novel combination of *dynamic mechanism design* with *stochastic planning* to tackle these challenges (Scharpff et al. 2013). Agents are rewarded or fined, according to the quality they deliver and the additional congestion caused by their activities on the network, such that in expectation their profit is maximal exactly when these global objectives are optimised. Here we study the application of that research with human players in a real-world setting.

Serious Planning Game

We have implemented the problem of maintenance planning and the solution we proposed in (Scharpff et al. 2013) in a serious simulation game, that we dub the serious planning game. Players, either humans or computer agents, take on the role of contractor and have to plan their maintenance activities such that their profits are maximised. They are supported by an automated planner that provides insight into payments and costs, and is able to provide plan suggestions, see Figure 2.

The major goals of our serious game are:

1. Studying whether our novel contracting method can be used in practical scenarios, and whether practitioners are likely to accept and adopt our method.
2. Creating awareness and support amongst practitioners regarding the impact of (coordinating) maintenance activities on a network level. Using this tool we want practitioners to get a feel for our novel and progressive concept, increasing the likelihood of acceptance.
3. Validation of the payment mechanism. Human players will most likely not be perfectly rational, therefore we study the strategies played by human planners and the resulting outcomes.
4. Closing the gap between theoretical concept and realistic contracting. This will increase the likelihood of practical implications.

In order to evaluate our serious planning game, we have developed questionnaires and observation protocols, allowing for a systematic analysis of the different problem factors.

Demonstration

We demonstrate the complexities faced in planning maintenance activities on a network and the need for a support tool through playing the game. Conference attendees will be given the possibility to participate in the game and experience the difficulty of finding (near-)optimal maintenance plans, while having to deal with other human or computer players. Games will be played through the use of our game interface, played on tablets, and participants will be asked to fill in short (simplified) questionnaires at the beginning and end of the game.

The main goal for a player is to plan his activities in the most profitable way. Activities can be performed in different (pre-determined) ways, varying in cost, duration, risk, quality effect and traffic disruption, and interfere with other player's maintenance. For instance, closing both the A97b and A101 of the network of Figure 1 concurrently causes major congestion while separate maintenance might introduce only little additional traffic hindrance. These situations are challenging and must be coordinated, either using the automated planner or by means of agreements through player-to-player communication. Eventually, players with unfinished tasks will be fined and the players that score best in each of the objectives (considering the portfolio it was given) are declared a winner.

References

Der Spiegel. 2012. A40: Autobahn nach dreimonatiger Sperre freigegeben. Online, Sep 30.

Scharpff, J.; Spaan, M. T. J.; Volker, L.; and de Weerdt, M. M. 2013. Planning under Uncertainty for Coordinating Infrastructural Maintenance. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*. To appear.

Volker, L.; Scharpff, J.; de Weerdt, M. M.; Herder, P. M.; and Smith, S. 2012. Designing a dynamic network based approach for asset management activities. In *Proc. of the 28th Annual Conf. of the Association of Researchers in Construction Management (ARCOM)*, 655–664.

Hypothesis Exploration for Malware Detection using Planning

Shirin Sohrabi, Octavian Udrea, and Anton V. Riabov
IBM T. J. Watson Research Center
PO Box 704, Yorktown Heights, NY 10598, USA
{ssohrab, oudrea, riabov}@us.ibm.com

Abstract[1]

In this paper we apply AI planning to address the hypothesis exploration problem and provide assistance to network administrators in detecting malware based on unreliable observations derived from network traffic. Building on the already established characterization and use of AI planning for similar problems, we propose a formulation of the hypothesis generation problem for malware detection as an AI planning problem with temporally extended goals and actions costs. Furthermore, we propose a notion of hypothesis "plausibility" under unreliable observations, which we model as plan quality. We then show that in the presence of unreliable observations, simply finding one most "plausible" hypothesis, although challenging, is not sufficient for effective malware detection. To that end, we propose a method for applying a state-of-the-art planner within a principled exploration process, to generate multiple distinct high-quality plans. We experimentally evaluate this approach by generating random problems of varying hardness both with respect to the number of observations, as well as the degree of unreliability. Based on these experiments, we argue that our approach presents a significant improvement over prior work that are focused on finding a single optimal plan, and that our hypothesis exploration application can motivate the development of new planners capable of generating the top high-quality plans.

[1]The paper is published in the Proceedings of the 27th Conference on Artificial Intelligence (AAAI-13).

Author Index