Planning to Plan — Integrating Control Flow

Alexander Nareyek

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3891, USA
alex@ai-center.com
http://www.ai-center.com/home/alex/

Abstract : In many planning situations, computation itself becomes a resource to be planned and scheduled. We model such computational resources as conventional resources which are used by control-flow actions, e.g., to direct the planning process. Control-flow actions and conventional actions are planned/scheduled in an integrated way and can interact with each other. Control-flow actions are then executed by the planning engine itself. The approach is illustrated by examples, e.g., for hierarchical planning, in which tasks that are temporally still far away impose only rough constraints on the current schedule, and control-flow tasks ensure that these tasks are refined as they approach the current time. Using the same mechanism, anytime algorithms can change appropriate search methods or parameters over time, and problems like scheduling critical time-outs for garbage collection can be made part of the planning itself.

Keywords : Meta-Planning; Resource-Bounded Reasoning; Online Planning; Integrated Planning, Sensing and Execution; Integrated Planning and Scheduling; Hierarchical Planning; Anytime Algorithms; Local Search; Metaheuristics

1 Introduction

Managing the control flow of a planner[1] is becoming an increasingly important part of the planning process:

- Online planning capabilities are in increasing demand to enable planners to respond quickly to *unforeseen problems* and to continuously adapt and optimize the plan to meet *changing requirements*. Computation power itself is becoming a resource here that must be planned and scheduled.
- Many planning decisions rely on information that will only become available at a later time. For example, how long a task needs to be executed or what price a negotiation will result in. Some *planning choices must be deferred* accordingly, and it may be necessary to trigger a replanning phase.
- Even during offline computations, directing the search via *anytime techniques* is often useful. Control-flow tasks like monitoring the state of the search and switching parameters or search methods accordingly must be handled in addition to the regular optimization process.

In this paper, we present an approach designed to merge planning of the control flow with the regular optimization process of the online, cost-based, integrated sensing, planning, scheduling and execution EXCALIBUR agent system [2, 6].

The general idea of integrated control-flow planning is, of course, not new. Early work on this topic was done with the action-planning system MOLGEN [10]. However, this approach took into account neither the online situation nor costs and resources. It was thus able to produce a feasible control flow for off-line planning but not to optimize it with respect to certain criteria or to handle the problems of an online situation.

Work on anytime procedures (see, e.g., [13]) focused on the restricted problem of trading computation time for solution quality, mostly with applications in the scheduling area. Instead of a general action-planning approach for planning the control flow, specialized method selections are applied here. Similarily, systems like MRG [12] only provide a scripting language for handling control flow, instead of automatic planning routines.

The approach presented here allows the system to apply its full planning capabilities to the control flow because planning and meta-planning are seamlessly integrated. Their tight integration also has the advantage that dependencies between control tasks and properties of the current and projected plan can directly interact within the optimization process.

Section 2 describes a simple planning model, which is used in Section 3 to show how integration of the control flow can be accomplished. A model extension, which is necessary for a more sophisticated control flow to handle decisions dependent on the current plan state, is described in Section 4. Conclusions are given in Section 5.

2 Model Structures

This section describes a simplified version of the integrated scheduling and action-planning model used for the EXCALIBUR agent's planning system [2, 6]. It is based on con-

[1] We use the word *planner* here as an umbrella term for scheduling as well as action-planning systems.

straint programming, i.e., *constraints* that enforce relations over *variables*.

Plan Structures and Variables

The model focuses on resources. A resource consists of a temporal projection of a specific property's state and a set of internal constraints, such as possible values and possible state transitions. A resource is also subject to further constraints induced by the plan, such as preconditions and state changes. Numerical as well as symbolic properties are uniformly treated as resources. For example, a production unit's TASK ASSIGNMENT, a battery's POWER and the state of a DOOR are all resources:

Resource TASK ASSIGNMENT:
  Internal constraints:
    { TASK ASSIGNMENT(t) ∈ { IDLE, TASK1, TASK2, TASK3} }
  Current projection:

$$\text{TASK ASSIGNMENT}(t) = \begin{cases} \text{TASK1} & : \ t \in [0..12] \\ \text{IDLE} & : \ t \in [13..16] \\ \text{TASK2} & : \ t \in [17..31] \\ \text{IDLE} & : \ t \in [32..\infty[\end{cases}$$

Resource POWER:
  Internal constraints:
    { POWER(t) ∈ [0..100] ,
      abs(POWER(t) - POWER(t+1)) < 10 }
  Current projection:

$$\text{POWER}(t) = \begin{cases} 0 & : \ t \in [0..5] \\ 10 - 0.75 \times t & : \ t \in [6..13] \\ 0 & : \ t \in [14..\infty[\end{cases}$$

Resource DOOR:
  Internal constraints:
    { DOOR(t) ∈ { OPEN, CLOSED, LOCKED, UNKNOWN} }
  Current projection:

$$\text{DOOR}(t) = \begin{cases} \text{OPEN} & : \ t \in [0..45] \\ \text{CLOSED} & : \ t \in [46..60] \\ \text{UNKNOWN} & : \ t \in [61..\infty[\end{cases}$$

An action (e.g., eating a peanut) consists of a set of different preconditions that must be satisfied (e.g., that the agent has a peanut), operations that must be performed (e.g., with the mouth) and resulting state changes (e.g., the agent's hunger being satisfied). These elements are represented by tasks, i.e., there are PRECONDITION TASKS for precondition tests, ACTION TASKS for operations and STATE TASKS for state changes. The tasks are *objects* — collections of *variables*, e.g., temporal variables determining the beginning and end of the tasks, or variables specifying how a state's properties are changed by the action. All tasks are assigned to resources — ACTION TASKS to ACTION RESOURCES, and PRECONDITION TASKS and STATE TASKS to STATE RESOURCES.

Figure 1 shows an example illustrating the planning model's basic elements.

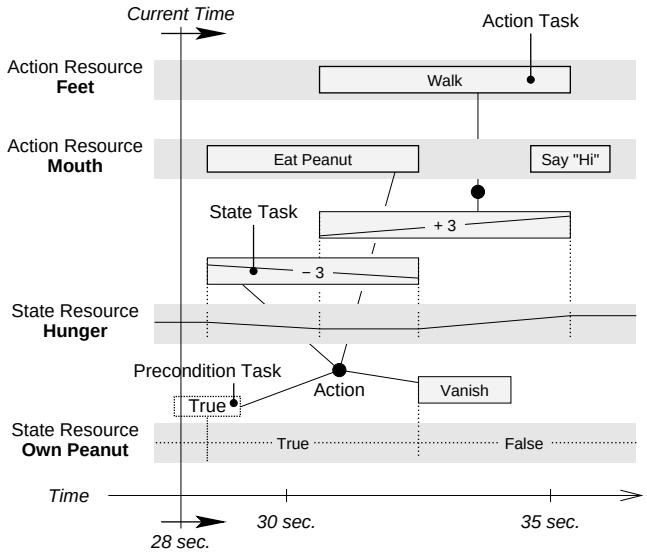

Figure 1: Plan Elements

Constraints

The validity of a plan is checked using *constraints*. A constraint is a test that checks a specific relation between certain variables or other model elements. For example, a constraint EQUALS can check the equality of two variables. If all of a problem's constraints are fulfilled, the model structures constitute a valid solution to the problem. For the planning model described here, there are three types of constraints:

- An ACTION RESOURCE CONSTRAINT (ARC) checks, for an ACTION RESOURCE, if there is enough capacity to carry out the operations, i.e., that the number of task overlaps at the resource does not exceed the resource's capacity (e.g., that the resource mouth, with a capacity of one, does not eat and talk at the same time).

- A STATE RESOURCE CONSTRAINT (SRC) checks, for a STATE RESOURCE, if the PRECONDITION TASKS of the resource are satisfied by the states deduced from the temporal projection of the resource's STATE TASKS (e.g., that the 'true' PRECONDITION TASK of the action eating a peanut is met).

- A TASK CONSTRAINT (TC) represents action requirements by specifying a relation between a set of PRECONDITION TASKS, ACTION TASKS and/or STATE TASKS. The constraint is satisfied if the tasks represent a valid action, i.e., if the correct tasks are involved and if they fulfill certain restrictions (e.g., that the PRECONDITION TASK and the ACTION TASK of the action eating a peanut begin at the same time).

When an 'action' is mentioned below, this corresponds to a TASK CONSTRAINT and the tasks associated with it.

Satisfaction/Optimization

Which techniques are used to solve/optimize a planning problem actually does not really matter here. We focus on a local search approach based on the DragonBreath engine [1, 5]. Local search means that there is always a complete assignment for the variables, and that this assignment is iteratively changed toward a satisfactory/optimal solution. The quality of a current solution is measured by a so-called *cost/objective function*. Unlike refinement search, the iterative repair/improvement steps of local search make it possible to easily interleave sensing, planning and execution. After each single repair step, which can usually be computed very quickly, the planning process can involve changes in the situation.

In our local-search approach, each type of constraint has an individual cost function. A constraint's cost function returns a value representing the constraint's current inconsistency/optimality with respect to the connected variables. For example, the cost function of the TASK CONSTRAINT might return a value of 0 if its tasks' temporal variables formed a required temporal relation, otherwise, a value that expresses the distance by which the variables' values would have to be shifted in order to satisfy the temporal relation. The cost functions of all constraints are combined to give an overall cost function (e.g., by simple addition or a more complex hierarchical mapping).

In addition, a constraint has *constraint-specific heuristics* that can change the values of the variables involved in order to improve the constraint's cost function value. For example, a constraint might decrease a task's *Begin* variable to move the task to a position with fewer overlap inconsistencies or to improve the constraint's goal, like an early completion time. In each improvement iteration, one of the unsatisfied constraints is called to improve its costs. However, the repair heuristics are not the focus of this paper; the interested reader is referred to [5] for details.

3 Integrating Control Flow

Control-flow tasks can be interpreted as regular tasks that are not assigned to regular resources but to a special control-flow resource. Unlike regular tasks, control-flow tasks are directly executed by the solver engine when they reach the current time and do not trigger external operations. Multiple computers/CPUs can be modeled via multiple control-flow resources.

Figure 2 gives some examples of control-flow tasks, e.g., one that monitors the progress of a task's execution to enable subsequent tasks to be preemptively shifted in the case of a likely delay. Another control-flow task is scheduled to reserve computational resources for a data backup.

Most search properties (like the current costs, the temperature in simulated annealing or the length of a tabu list) and their adaptation will be kept inside the optimization routine. But sometimes it is useful to explicitly model this state such that it can be involved into the planning process, e.g., if transmission of the plan to one of the machine groups

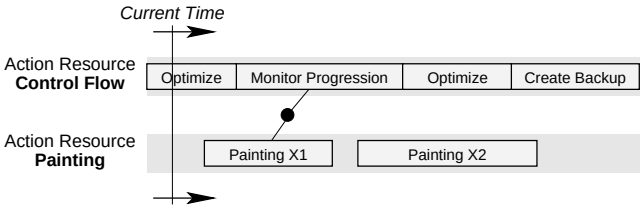

Figure 2: The Control-Flow Resource and Tasks

for execution is dependent on the non-existence of critical overlaps of tasks (i.e., all related ACTION RESOURCE CONSTRAINTs having costs of 0). The anticipated cost development for the overlaps can be projected by a `State Resource Constraint` and tested using a PRECONDITION TASK for the transmission action (see Figure 3).

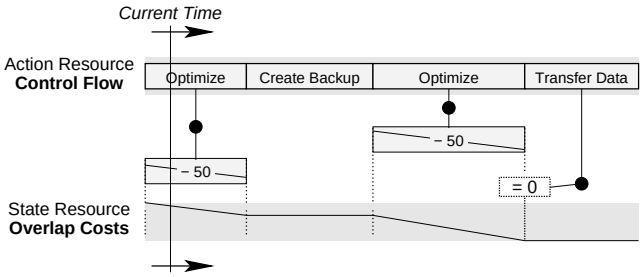

Figure 3: Explicitly Modeling Search Properties

Optimization as a Default Action

Handling control-flow tasks/resources requires a slightly different treatment than the regular ones. First of all, optimization is one of the main tasks to be planned/executed by control-flow resources, and it would not be good if control-flow resources remained idle rather than further optimizing the current solution. For this reason, an *optimization is always inserted as a default action*.

Percentage-Wise Resource Consumption

There are some tasks that require attentiveness over a lengthy time period, e.g., to monitor certain conditions. Such tasks can usually be executed very fast, but an excessive number of them would be needed to cover a lengthy time period. Thus, we allow tasks to be permeable to other operations, i.e., *every control-flow task specifies for which percentage of the task's duration its operations should be executed* (its operations being uniformly distributed over the task's duration). The remaining capacity can be used by other tasks or the default optimization task.

Reservations for Control-Flow Improvement

Because of local search's iterative repair approach, it may happen that control tasks are placed inconsistently, e.g., if

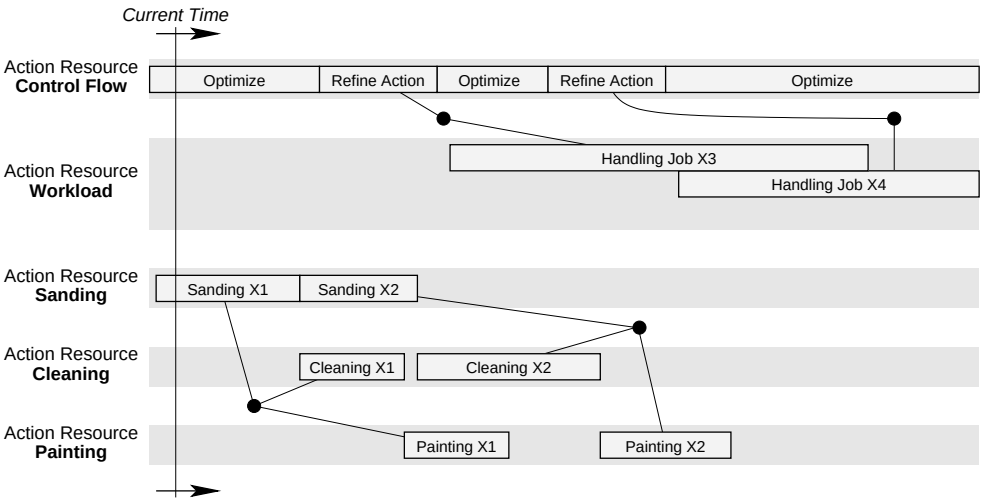

Figure 4: Hierarchical Online Scheduling

overlapping tasks were to consume more than 100 % of the resource, but the control-task resource's schedule has no task for optimization scheduled until then. Situations like this must be averted. This can be done by *reserving a certain percentage of the control-task resource for control-task-related cost improvement* at any time.

An Application Example: Hierarchical Planning

In this section, we give an example of how control-flow tasks can be used in online job-shop scheduling. One critical problem in an online domain is that computational power is highly restricted and the problem situation usually changes frequently, e.g., because of new, canceled or tardy jobs.

Let there be n jobs $j^1 ... j^n$ in a schedule, each of them representing an action with m ACTION TASKs $t_1^j ... t_m^j$ with fixed given durations. All tasks of a job are connected via $m-1$ linear distance inequalities involving start or end time points of two tasks. To comply with jobs' execution deadlines $d^1 ... d^n$, there is in addition a linear distance inequality $t_i^j(end) \leq d^j$ for each task i of a job j. The temporal relations of a job/action are enforced via a TASK CONSTRAINT.

Taking care of all constraints is not easy — at least for large-scale problems with fast job throughput. The idea, then, is to have the planner create a rough plan first, and as the abstract tasks get closer to execution, refine it stepwise toward a fully specified plan. Related work includes [4] in the scheduling area and [7, 8, 9, 11] in the planning area (although the latter approaches do not consider the temporal refinement aspect).

This technique is very useful because it gives the planner the ability to take a high-level view and potentially saves handling low-level details for distant tasks for which the requirements are still likely to change.

The procedure can easily be realized using our control-flow concept. Abstract tasks and constraints are first introduced, then refined by control-flow tasks that transform the structures into more detailed levels. In general, the way the refinement should take place is highly dependent on which kind of constraints play the most important role. For the example shown in Figure 4, a general WORKLOAD resource is introduced, on which abstract tasks HANDLING JOB XJ are placed that represent a cumulative workload for all tasks of a job[2].

The duration of an abstract task t_0^j is set as the added duration of the job's detailed tasks $t_1^j ... t_m^j$ plus an experience-based slack time. There is also a linear distance inequality $t_0^j(end) = d^j$. The capacity of the WORKLOAD resource is set to the number of jobs that can usually be processed in parallel (3 in the example figure). When a control-flow task for refinement is executed, the task itself and the related abstract task t_0^j are replaced by $t_1^j ... t_m^j$, and the new tasks are randomly distributed over the duration of t_0^j at their respective resources.

A tighter integration of the different hierarchies is also possible, e.g., allowing a refinement to occur only if there are no overlaps at the resources at this time (as for the action of data transfer in Figure 3). Many other refinement options can be modeled, and the demonstrated solution is probably not the optimal one. The aim here was to show how control-flow tasks can be used for this purpose.

4 State-Dependent Decisions

The first application area of the control-flow integration in our EXCALIBUR project will be for negotiation tasks (see Figure 5). For tasks like negotiation, many different refinements may exist. This can easily be modeled, as multiple possible refinements can be represented using an abstract

[2]The control-flow ACTION TASKs to refine the actions have been extended for the sake of clarity. Normally, this should take only one iteration.

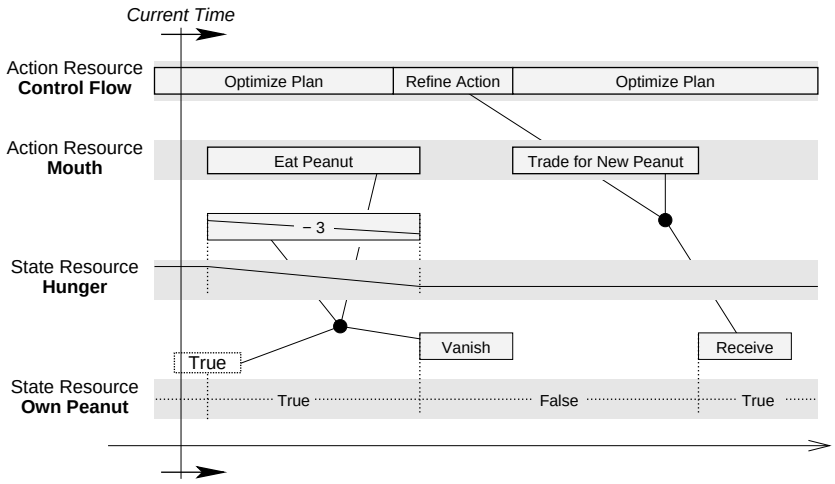

Figure 5: Delaying the Refinement of a More Complex Trading Action

task. The procedure of choosing between alternative refinements can be integrated into the task's execution operation, potentially being dependent on other states of the current plan.

Providing a Tool for Reading a Specific State

If an operation is to be dependent on a state of the current plan, the operation's ACTION TASK needs to have a temporally qualified link to the respective STATE RESOURCE to identify which value is to be checked. This is similar to a PRECONDITION TASK, except that it is not a comparison test for a specific value but a reading operation. The planning model is therefore extended by READING TASKs, which enable STATE RESOURCES' values to be accessed at a specific time or over a time interval. A READING TASK is part of an action, and its variables are related to the other action tasks via the action's TASK CONSTRAINT. Figure 6 shows an example of a READING TASK for the trading action, which allows the refinement operation to refine the trading action depending on the location (e.g., if the agent is in a shop or in an auction house).

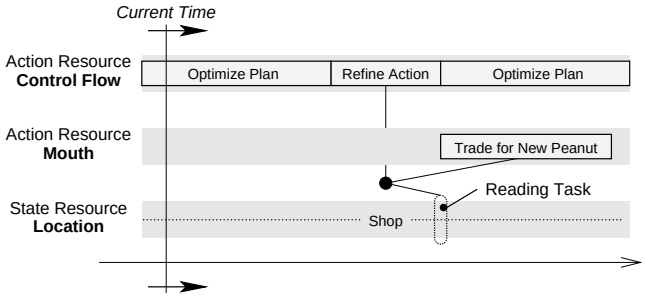

Figure 6: Reading Tasks

A special mechanism for writing to a STATE RESOURCE is not necessary as this can be handled by STATE TASKs.

An Application Example: Evaluating Alternatives

In some situations, it is useful to explicitly evaluate specific alternative plans, e.g., to calculate a worst-case resource consumption or to compare specific features of plan alternatives. This can be interpreted as a kind of advanced neighborhood analysis and provides a very powerful control-flow tool.

In the example of the trading action, one may want to evaluate via refined tasks whether one should accept an offer or not. The trading action can expand to a group of more detailed control-flow tasks that evaluate these alternatives (see Figure 7).

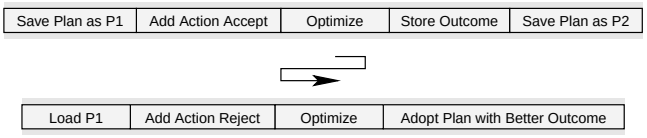

Figure 7: Control-Flow Tasks for Evaluating Alternatives

The control flow shown in Figure 7 must make further use of hierarchical refinement for the operation ADOPT PLAN WITH BETTER OUTCOME (possibly loading plan P2), and applies READING TASKs for STORE OUTCOME (reading the current costs) and ADOPT PLAN WITH BETTER OUTCOME (reading the stored and current costs).

In parallel to the tasks of Figure 7, another control-flow task must keep track of external situation changes and either abort the evaluation of the alternatives in the case of critical changes or store them for later adoption once the plan with the better outcome has been chosen.

If there are many alternatives and not enough time available to evaluate all of them, it might be a good idea to add a hierarchy layer to schedule the sequence of evaluations according to the most promising ones or to decide which subset of evaluations should be executed. For a specific schedul-

ing domain, related work can be found in [3].

5 Conclusion

We have presented a concept for a tight integration of control flow and the regular optimization process. The planning of the control flow can make full use of the system's automatic planning capabilities, and control-flow actions and regular actions can directly interact with each other. Possible applications include garbage collection, backups and data transfers that can be planned depending on the optimization state, integration of hierarchical action transformations, and the evaluation of alternatives. In addition, this allows search mechanisms that are tightly coupled to the search state — like exploration/diversification and exploitation/intensification — to be scheduled in accordance with specific search-state features.

Our future work will make extensive use of these concepts, especially for combining planning and negotiation. More information on the underlying EXCALIBUR project is available on the project's website [2].

References

[1] The DragonBreath engine's homepage: http://www.ai-center.com/projects/dragonbreath/

[2] The EXCALIBUR project's homepage: http://www.ai-center.com/projects/excalibur/

[3] Garvey, A. J. 1996. Design-To-Time Real-Time Scheduling. PhD thesis, University of Massachusetts, Amherst MA.

[4] Marx, P. 1994. Refinement Scheduling — Verfahren zur Organisation komplexer Arbeitsvorgänge. Diploma Thesis, Department of Computer Science, Technical University of Clausthal, Clausthal-Zellerfeld, Germany.

[5] Nareyek, A. 2001. Using Global Constraints for Local Search. In Freuder, E. C., and Wallace, R. J. (eds.), *Constraint Programming and Large Scale Discrete Optimization*, American Mathematical Society Publications, DIMACS Volume 57, 9–28. Available via: http://www.ai-center.com/references/nareyek-01-cplocal.html

[6] Nareyek, A. 2001. *Constraint-Based Agents – An Architecture for Constraint-Based Modeling and Local-Search-Based Reasoning for Planning and Scheduling in Open and Dynamic Worlds*. Reading, Springer LNAI 2062. Online content via: http://www.ai-center.com/references/nareyek-01-planheuristics.html

[7] Nau, D.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple Hierarchical Ordered Planner. In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99), 968–973.

[8] Rabideau, G.; Knight, R.; Chien, S.; Fukunaga, A.; and Govindjee, A. 1999. Iterative Repair Planning for Spacecraft Operations in the ASPEN System. International Symposium on Artificial Intelligence Robotics and Automation in Space (iSAIRAS 99).

[9] Sacerdoti, E. D. 1974. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence* 5(2): 115–135.

[10] Stefik. M. J. 1981. Planning and Meta-Planning (MOLGEN: Part 2). *Artificial Intelligence* 16(2): 141–169.

[11] Tate, A. 1977. Generating Project Networks. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77), 888–893.

[12] Traverso, P.; Cimatti, A.; and Spalazzi, L. 1992. Beyond the Single Planning Paradigm: Introspective Planning. In Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92), 643–647

[13] Zilberstein, S. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine* 17(3): 73–83.

Alexander Nareyek studied computer science at the TU Berlin, from which he also received his Ph.D. He won the GMD's 1997 Best Degree Thesis Award, was a winner of the BMWi's Founders Competition Multimedia 1998 and was awarded an enhanced Ph.D. scholarship by the German Research Foundation (DFG). He is involved in many AI events, e.g., as organizer of several workshops on planning topics and as chairperson of the IGDA's AI Interface Standards Committee. Since 2002, he has been guest researcher at CMU on an Emmy Noether fellowship.