Old and New Ideas for Integrating Planning and Execution

Othar Hansson, Jordan M. Hayes, Charles A. Ocheret

Thinkbank, Inc.
1678 Shattuck Avenue, Suite 320, Berkeley, CA 94709-1631
othar@thinkbank.com, jordan@thinkbank.com, chuck@thinkbank.com

Abstract

We sketch some features of the planner/executive interface in SchedKit, a toolkit of reusable software components for science planning and spacecraft sequencing. Our interest in execution was spurred by lessons from the New Millenium Remote Agent (NMRA) project. Many other computer scientists have also studied the control of execution in complex, uncertain environments: here we describe elementary concepts of *transaction-processing* that have inspired our design. Finally, we describe some novel modeling techniques supported by SchedKit.

Planning and Execution

Thinkbank, Inc. is developing SchedKit, a toolkit of reusable software components for science planning and spacecraft sequencing. This paper describes the planner/executive interface within SchedKit. Our design has been influenced by the New Millenium Remote Agent (NMRA), currently being implemented at NASA Ames and at JPL for the DS-1 asteroid rendezvous mission. Details of NMRA can be found in recent papers by Pell et al. (96a,96b,97a,97b) and Muscettola et al. (97).

We are particularly interested in the interface between the planner and the executive. How does the planner plan for execution? How should the two modules divide up the labor: how much reasoning and constraint processing should be done at run-time? What "virtual machine" should the executive present to the planner? Is there a single representation of plans that meets the needs of planning and execution?

Historically, planners and schedulers have not taken seriously the problems of execution. This dates back to the earliest efforts at optimizing operations. For example, one of the 19th-century ancestors of modern operations research was Frederick Taylor: "Taylorism" or scientific management was rightly criticized for placing all the focus on planning and scheduling (i.e., management functions), and for assuming that flawless and repeatable execution (by human laborers) was possible. The vain attempt to separate cognition from action is recapitulated in modern operations research and artificial intelligence. As a result, the brittleness of schedules still limits the adoption of scheduling technology in factories, transportation systems, space missions, and other application domains.

Much recent work in AI has attempted to integrate planning and execution. And the NMRA team has highlighted the importance of integrating planning and execution, if a scheduler is to be flown on-board a spacecraft: in fact, Pell et al. (97a) claim that "NMRA is one of the first systems to integrate closed-loop planning and execution of concurrent temporal plans."

NMRA consists of three main modules: The Planner/Scheduler (PS), the Executive (EXEC) and the Mode-Identification and Recovery system (MIR). Loosely speaking, PS generates episodic plans, passes them to EXEC for execution, and waits for the planning horizon to expire or for MIR or EXEC to detect failure conditions. MIR has its own lower-level rapid recovery planner as well. MIR and EXEC handle some errors, but when a plan failure occurs, PS is reinvoked to generate a plan to patch in at a rendezvous point in the future.

In theory, PS can produce more robust schedules than traditional schedulers, because PS produces least-commitment schedules, with flexible time-windows for each spacecraft activity. That this is a revolutionary idea is perhaps reflected in the widespread use of "sequence" to describe the output of NASA schedulers and planners: a sequence is a total order, but PS produces plans that are partial orders with some run-time flexibility. In other words, each activity is not assigned a precise time for execution: the executive can choose a time consistent with a set of simple temporal constraints (difference constraints). At the expense of added software complexity, this least-commitment approach should allow execution to continue even after deviations from predicted behavior.

The Planning/Execution Interface: PRL

In designing SchedKit, we have the luxury of starting with a clean sheet of paper in designing the planner, the executive, and the interface between them. In contrast, NMRA began with relatively standard architectures for the planner (based on HSTS) and the executive (based on RAPS), and designed the planner/executive interface by adjusting existing inputs and outputs. This was of course necessary to meet mission schedules and reduce risk.

If, however, we start from scratch, one way to approach the design problem is to pursue an "execution-as-computation" or "computer language" analogy.

Specifically, the plan representation language (PRL) is viewed as the instruction set of a virtual machine; the executive is the runtime system for this virtual machine; and the planner is the compiler. The planner compiles sets of goals (the source language) into plans (the target language, i.e., the plan representation language), which are then executed (by the executive, which implements the target language virtual machine). The activities in the plans are operations made available by the virtual machine.

There are some notable constraints to the design problem. The planner's inputs (the source language) are constrained by the demands of the application. In the case of a spacecraft, the source language must be simple but flexible to meet the demands of ground controllers. Likewise, the executive's outputs are tied to the spacecraft hardware. But importantly, the PRL interface between planner and executive is unconstrained.

There are many choices in designing the PRL. By making the virtual machine more powerful, we can replace the executive's behavior and logic by corresponding statements in the plan representation language. Conversely, we can make the plan representation language trivial, i.e., equivalent to the source language, thus passing all of the planner's "normal" responsibility onto the executive. This is a well-examined tradeoff in the design of computer instruction sets ("reduced" versus "complex" instruction sets, i.e., RISC versus CISC). A low-level PRL (analogous to a RISC design) allows for a very simple and fast executive, but requires more work on the part of the planner (or compiler).

The PRL choice in NMRA was made according to the following principle: "In DS1, activities are abstracted to the level where there are no interactions among their subactivities. This level allows the planner to resolve all of the global interactions without getting into details that would over-constrain the executive." (Pell et al. 97b). Over-constraining EXEC would increase the risk of plan failure. PRL should also specify failure modes and recovery alternatives: in NMRA, recovery semantics seem to be represented entirely within EXEC, and not within the "plans" themselves.

Virtual Machine example: Transactions

In designing SchedKit, we sought to borrow as much technology as possible from other areas of computer science. Transactions are a central abstraction used in implementing modern distributed systems. As Gray and Reuter (93) put it, "In a nutshell: without transactions, distributed systems cannot be made to work for typical real-life applications." And yet transactions arose not as a naturally evolved abstraction or requirement, but as a radical proposal to overcome the chaos of distributed systems implementation projects at IBM and elsewhere.

The "ACID" properties define transactions:

- Atomicity

  transactions have "all-or-nothing" effects, and will undo on abort.
- Consistency

  integrity constraints are maintained.
- Isolation

  each transaction executes as if unaffected by concurrently executing transactions.
- Durability

  after a system failure, the achieved state of the system is recovered automatically.

Through the ACID properties, a database management system provides a virtual machine: in other words, it guarantees certain behavior that is abstracted away from the uncertainty or other details of actual execution.

We will briefly compare transaction-processing applications to "typical" planning/scheduling systems. Both need atomicity because failures are unavoidable, and application designers demand some simple and universal recovery mechanism. Ad hoc recovery mechanisms, with unique guarantees, result in chaos.

Durability deals with even more sophisticated error recovery: ensuring that the achieved state of the system is re-created after a failure (part of MIR's role). But in a planning/scheduling application, some aspects of the system state may not be controllable, while others may be irrelevant to the remainder of the plan. In short, only the preconditions for the remainder of the plan need to be durable in the face of failures. Finding such a satisfactory recovery state after failure may involve search. But without some durability mechanism, execution halts at the first global error that was not anticipated in the plan itself (e.g., as a contingent execution path).

Consistency is a responsibility that is presently diffused among the plan itself, EXEC, and MIR: we advocate collecting and exposing these integrity constraints. The final ACID property is isolation: activities may be easier to specify if we can isolate them from the immediate effects of concurrently-executing activities. Isolation is partially achieved in planner/schedulers by careful specification of preconditions and "parameter bindings" within activities. After parameters are bound and preconditions checked, the activity is typically oblivious to changes in parameters and preconditions.

In some senses, the demands of planning and scheduling are more difficult than transaction-processing. But if our systems lack the fundamental ACID properties such as atomicity and consistency, then it will be as difficult to debug planning applications as it is to debug distributed systems without transactions. It may be that entirely different abstract properties or "guarantees" will prove to be more suitable. But the ACID properties are a natural starting point, and we hope to prove or disprove their usefulness in the applications we prototype and develop using SchedKit.

Execution Support in SchedKit

A few of SchedKit's features are directly borrowed from the transaction-processing framework (Gray and Reuter 93, Papadimitriou 86). By using these familiar guarantees provided by transaction-processing systems, we hope to simplify the task of designing and debugging planning/scheduling applications. Still other features are borrowed from the distributed systems "protocol design and validation" literature (e.g., Holzmann 91).

Activities in a SchedKit plan are atomic in the sense that failure modes and recovery actions can be specified. Failure modes are simply represented as preconditions for the corresponding recovery actions. Global plan failures can be monitored by activities inserted into the plan. These failure models are incorporated in the plan by use of a simple task decomposition (i.e., they are automatically included as part of the expansion of a higher-level activity). Consistency of global integrity constraints can be checked similarly: an integrity constraint is an activity that is always ready-to-run, but whose preconditions are triggered only when an integrity constraint is violated. Finally, as a durability mechanism, we hope to be able to use an existing recovery system.

One major gap that we see in SchedKit and NMRA (and other systems) is a lack of a metric for execution performance. What exactly is the goal of the executive in a particular domain? An executive which "fails aggressively" produces safe but unproductive results. An executive which "recovers aggressively" risks spacecraft health. We feel that such tradeoffs can only be addressed by explicit modeling: each failure and recovery should somehow be annotated with its associated risks. Utility theory is one promising modeling tool available to place this planning/execution tradeoff on a sound footing. We hope to extend SchedKit's modeling language in this direction, based on some of our previous decision theory work (Hansson 97).

One final point on SchedKit's design is the use of parameterization throughout the system. Wherever possible, decisions are reified or represented as explicit constrained variables. For example, in modeling the constraints of the domain (analogous to NMRA's Domain Description Language), we use Modeling variables and named constraints to make design assumptions explicit ("if Modeling(EffectX) then Enforcing (ConstraintY)"). We are experimenting with a similar approach to represent abstraction levels. A final example of parameterization is within the search algorithms: branch points, algorithm parameters, etc. are reified as variables to aid in explanation.

For further details on SchedKit, visit our web-site at http://www.thinkbank.com.

Acknowledgments Thanks to many members of the NMRA team at NASA Ames and JPL for helpful discussions. This work is supported by NASA Contract NAS2-97008.

References

[Gray and Reuter 93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, San Mateo, CA.

[Hansson 97] Othar Hansson. Bayesian Problem-Solving Applied to Scheduling. Doctoral dissertation, University of California (Berkeley), forthcoming.

[Holzmann 91] Gerald J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, Englewood Cliffs, NJ.

[Muscettola et al. 97] Nicola Muscettola, Chuck Fry, Kanna Rajan et al. On-Board Planning for New Millenium Deep Space One Autonomy. In *Proceedings of IEEE Aerospace Conference*, Snowmass, CO.

[Papadimitriou 86] Christos Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Press, Rockville, MD.

[Pell et al. 96a] Barney Pell, Douglas E. Bernard, Steven A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams. A Remote Agent Prototype for Spacecraft Autonomy. In *Proceedings of the SPIE Conference on Optical Science, Engineering, and Instrumentation.*

[Pell et al. 96b] Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith. Plan Execution for Autonomous Spacecraft. Appears in the *Working Notes of the AAAI Fall Symposium on Plan Execution,* Cambridge, MA, 1996.

[Pell et al. 97a] Barney Pell, Douglas E. Bernard, Steven A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams. An Autonomous Spacecraft Agent Prototype. In *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA 1997.

[Pell et al. 97b] Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith. Robust Periodic Planning and Execution for Autonomous Spacecraft. *Proceedings of IJCAI-97.*