Commentary on: Mission Planning and Execution Within the Mission Data System by A. Barrett, R. Knight, R. Morris and R. Rasmussen

Maria Fox
University of Strathclyde
Glasgow, UK

The agenda of this work is to manage the uncertainty inherent in the execution environment of an autonomous robot system such as a Mars rover. The approach moves away from a model in which action sequences are constructed and then executed, towards one in which reasoning and execution are seamlessly interleaved to support online integration of planning, scheduling and control. To achieve this integration the authors abandon the traditional use of different representation languages for reasoning and for execution. Instead, they adopt a uniform language for expressing state variables and their interactions. All conceptual objects (goals, constraints, actions etc) are then expressed in terms of temporally constrained state variables. The argument is that within this uniform framework control can be naturally interleaved with the satisfaction of constraints, allowing the system to perform locally fault-tolerant behaviours within the broader context of achieving its overall goals.

The main representation component of the MPE system is a *task network*. It consists of inter-dependent state variables, and constraints on their interactions over time. Goals are interpreted as requirements to have certain variables constrained in certain ways over certain time intervals. Actions correspond to sending constraints to the controllers associated with the relevant state variables. A task network can be incomplete in that the variable constraints involved in a particular goal are unsupported and require elaboration. Also a task network can be partially executed, since when a goal is fully elaborated the part of the network that enforces it can be scheduled for execution. In this framework, planning corresponds to the task of goal elaboration. Scheduling corresponds to the addition of temporal constraints to the network. Execution corresponds to the interpretation of part of a complete task network by the state variable controllers.

The language chosen for implementing this uniform representation is C++. The MPE system is implemented using base classes to represent the fundamental ideas of state variables, goals and other network structures, and inheritance to enable the specification of particular types of state variables. The reasoning components of the MPE are implemented as functions of these classes. Thus, the uniform representation formalism translates into a single, purpose built program for implementing the MPE.

The architecture proposed in this paper is extremely elegantly conceived. Of the many interesting and controversial issues it raises there are four which I will address in this commentary:

1. The idea of a uniform representation language in preference to languages purpose-built for particular reasoning tasks;
2. The use of C++ (or an equivalent) language for implementing the MPE + domain as a single program;
3. The focus on locally fault-tolerant behaviours as a way to direct the achievement of a complex global goal;
4. The likely utility of the proposed architecture in terms of the originally stated agenda of the paper.

The argument that one gets increased flexibility and/or increased efficiency when different problems are flattened into the same representation formalism is not immediately convincing. As an illustration, it is known that the Bin-Packing problem can be expressed in the language of Satisfiability (and vice versa) since they are both NP-complete, but it is also known that good approximate solutions to Bin-Packing instances can be found by exploiting the inherent structure of the Bin-Packing problem. There seems no good reason to believe that losing the structure that distinguishes the planning part of a problem from the scheduling part, or from other (possibly combinatorial) sub-problems is likely to increase efficiency in the solution of the overall problem. Indeed, flattening several structurally different problems into one formalism can result in a significantly larger problem representation, and a correspondingly larger search problem. Whilst theoretically the complexity of the search problem does not increase it might become less practical to solve.

Indeed, an orthogonal approach to this one is to decompose a problem around the structurally different subproblems it contains and to integrate specialised solvers for these sub-problems. Each solver solves its own sub-problem within the global context and their solutions must be combined into a solution to the global problem. There are many difficulties involved in communicating constraints between the sub-solvers and in ensuring the coherence of the global solution. Here the emphasis is on exploiting structure rather than flattening it, in the recognition that generic techniques are rarely as efficient as special purpose ones. After all, years of research have been invested in the solution of certain combinatorial problems and it seems unnecessarily reductionist to tackle such problems with brute force search.

Whilst it might be true that the use of a uniform representation makes it straightforward to communicate between execution and elaboration, the complexity of the management of this communication – the need to maintain the executing network, the copy, the links from the copied goals to the elaborators of the corresponding original goals and the need to manage the search problem that results – all suggest that in practice this management, though ingenious, is no less complex than the integration of different formalisms in a traditional multi-layered architecture.

The paper proposes the implementation of the MPE subsystem of the MDS, together with a domain specification, as a single C++ program. On one hand one can argue that C++ provides all of the benefits of encapsulation, modularity and abstraction available to the engineering of a large and complex software system, and that a complex planning/scheduling domain requires these tools for efficient representation. These are tools which are not yet available in the languages commonly used for building planning domains. Thus, C++ can provide a powerful modelling language as well as an expressive means of articulating the complexities of the domain. Furthermore, by providing a core set of base classes and requiring subclasses and instances to be defined as part of an explicit specification of a domain, the representation of the domain can, in principle, be kept separate from the algorithms used to reason about specific state variable types and constraints. This means that the generic components of the system can be reused.

On the other hand one can argue for the decomposition of a search problem into the domain file and the problem solver, on the basis that this allows for independent validation of the domain specification as well as independent debugging and optimization of the generic components. Strong coupling of components is always a structural weakness in software engineering designs. When domains are very complex the need for abstraction to facilitate the modelling and communication of domain behaviours, and of independent validation to confirm that the model is correct, become important. There is no guarantee that authors of C++ programs will use the facilities of abstraction in a way that usefully achieves abstraction, or that the resulting program will be modular or amenable to separate domain validation.

But this is to some extent a religious debate: researchers have been arguing for a long time about what kind of language and architecture is most appropriate for the modelling of complex planning and scheduling problems (and problems in general). Given the agenda specified at the beginning of this paper, the real question is: how likely is the proposed MPE architecture to increase the efficiency and reliability of spacecraft operations?

The key argument presented by the authors in support of their claim is that because control and reasoning can be integrated online the system can behave in a locally fault-tolerant way to achieve its global goals. Although this might be the case when the system can manage the complexity of its search problem (ie: for small problem instances in which the goals are not very complex) in general I doubt that a complex collection of interacting global goals is likely to be achieved on the basis of local behaviours. The greedy approach that is taken to alleviate the complexity of the integration between execution and elaboration means that there will sometimes be no local solution to the execution problems that arise and the system must enter a safe state. Of course, this would also be the case for an action-based system using a greedy search technique. It is known that greedy local search is not sufficient for solving arbitrary search problems, so why should flattening the representation make it easier to intelligently prune the search involved in integrating planning and execution?

Several themes are indicated by the authors as requiring further work. One of these is the management of continuous resources. In order to reason about how the achievement of particular goals can affect other aspects of system state it will be necessary to reason about continuous change. For example, if the consumption of power by the heater affects the ability of a transmitter to send data, then the system will need to be able to calculate the exact time at which the power level has fallen too low for the data to be transmitted and it will need to schedule the warming and transmitting actions accordingly. This becomes very complicated when there are multiple (non linear) interactions. In the context of a single C++ program implementing MPE + domain, problem specific hacks can be implemented to resolve specific continuous interactions. However, this approach would undermine the generality of the system so it remains to be seen whether such interactions can be managed in a satisfactory way.

This paper describes an elegant and appealing approach to the integration of planning, scheduling and execution. The search control mechanisms proposed are interesting solutions to the management of open search paths in the context of a partially executed task network. If rather controversial in it philosophy, the work makes an undeniably interesting contribution to research in plan execution in dynamic environments. What this paper lacks is evidence that local fault tolerant behaviour, made possible by means of a uniform representation enabling the direct integration of planning, scheduling and control, can be coupled with improvements in reliability and consequently, as the authors claim at the outset, a less stressful approach to the management of execution-time uncertainty.