processes, which interrelates states and maps states to observables. Finally, a reward function is associated with each automaton.

Constraint-based Trellis Diagram. Mode estimation encodes Probabilistic Hierarchical Constraint Automata (PHCA) as a constraint-based trellis diagram, and searches this diagram in order to estimate the most likely system diagnoses. This encoding is similar in spirit to a SatPlan/Graphplan encoding in planning.

Conclusion

We have extended a system capable of diagnosing and reconfiguring redundant hardware systems so that instrumented software systems can likewise be made robust. Software systems lack many of the attributes of hardware systems to which the described methods have traditionally been applied; they tend to be more hierarchical and have more complex and numerous component interactions. Software components and their interconnections represent a significantly higher modeling burden.

Our approach differs from other similar techniques in the following ways:

- Models specify program behavior in terms of abstract states, which simultaneously makes the models easier to read and think about and somewhat robust to changes in low-level software implementation decisions.
- Modeling covers a wide spectrum of software considerations from a high-level storyboarding of the software to temporal considerations, if any, to the causal relationships between components.
- Robustness and recovery derives from a collection of complex and highly tuned reasoning algorithms that estimate state, choose contingencies, and plan state trajectories. The programmer is largely shielded from this complexity because the mechanism is hidden behind the intuitive unified modeling language.

An interesting feature of our approach is the ability to add robustness incrementally. More modeling leads to greater runtime robustness because it allows the system to detect, diagnose, and repair more fault situations. This means the effort devoted to modeling can be managed in much the same way that is done for test suite development in conventional software development projects.

Modeling errors can result in a number of undesirable outcomes such as failure to detect fault conditions and subsequent failure to recover from the fault, incorrect diagnosis of the fault, and attributing faults to components that are operating correctly. In this sense an incorrect model is no different from any other bug in the software. It is somewhat easier to deal with, however, because the models are written at a more abstract level than the program itself, making them easier to read. There is a problem with making changes to the software definition and neglecting to update the models of the software. In time we expect tools to evolve to address this kind of problem.

The nature of the models developed for a software system vary depending upon the nature of the software itself. Some programs, especially those involved in embedded and robotic applications, have critical timing considerations that must be modeled as such whereas other programs have no timing of synchronization considerations.

Developing model-based reconfigurable software systems is a relatively new endeavor, but results of our early experiments are encouraging. Much work remains to extend the current experimental system to cover the full range of software practice. C

References

1. Bernard, D., Dorais, G., Gamble, E., et al. Spacecraft autonomy flight experience: The DS1 remote agent experiment. In *Proceedings of the AIAA Space Technology Conference and Exposition* (Albuquerque, NM, Sept. 1999).
2. Firby, R. *The RAP Language Manual.* Working Note AAP-6, University of Chicago, 1995.
3. Gat, E. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Proceedings of the AAAI Fall Symposium on Plan Execution* (Cambridge, MA, Nov. 1996).
4. Laddaga, R., Robertson, P., and Shrobe, H.E. Introduction to self-adaptive software: Applications. In *Proceedings of the 2nd International Workshop on Self-Adaptive Software (IWSAS 2001),* (Balatonfüred, Hungary, May 2001), LNCS 2614, Springer.
5. Mikaelian, T., Williams, B.C., and Sachenbacher, M. Diagnosing complex systems with software-extended behavior using constraint optimization. In *Proceedings of the 16th International Workshop on Principles of Diagnosis (DX-05),* (Monterey, CA, June 2005).
6. Simmons, R. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation 10,* 1 (1994), 34–43.
7. Williams, B. and Nayak, P. A reactive planner for a model-based execution. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97),* (Nagoya, Japan, August 1997).

Paul Robertson (paulr@csail.mit.edu) is a research scientist at the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory, Cambridge, MA.
Brian Williams (williams@mit.edu) is Boeing Associate Professor of Aeronautics and Astronautics at the Massachusetts Institute of Technology, Cambridge, MA.