the service model through the inclusion of unmodeled failure modes. It is used by a deductive controller to map sensed variables to queried states. The service model is specified as a concurrent transition system, composed of probabilistic concurrent constraint automata. Each component automaton is represented by a set of component modes, a set of constraints defining the behavior within each mode, and a set of probabilistic transitions between modes. Constraints are used to represent co-temporal interactions between state variables and intercommunication between components. Constraints on continuous variables operate on qualitative abstractions of the variables, characterized by the variable's sign (positive, negative, zero) and deviation from nominal value (high, nominal, low). Probabilistic transitions are used to model the stochastic behavior of components, such as failure and intermittency. Reward is used to assess the costs and benefits associated with particular component modes. The component automata operate concurrently and synchronously.

Self-Deprecation and Regeneration through Predictive Method Dispatch. In model-based programming, the execution of a method fails if one of the service components it relies upon irreparably fails. This in turn can cause the failure of any method that relies upon it, potentially cascading to a catastrophic and irrecoverable systemwide malfunction. The control sequencer enhances robustness by continuously searching for, and deprecating, any requisite method whose successful execution relies upon a component that is deemed faulty by mode estimation, and deemed irreparable by mode reconfiguration.

Without additional action, a deprecated method causes the deprecation of any method that relies upon it. Model-based programmers specify redundant methods for achieving each desired function. When a requisite method is deprecated, the control sequencer attempts to regenerate the lost function proactively by selecting an applicable alternative method, while verifying overall safety of execution.

More specifically, *predictive method selection* first searches until it finds a set of methods that are consistent and schedulable. It then invokes the dispatcher, which passes each activity to the deductive controller as configuration goals, according to a schedule consistent with the timing constraints. If the deductive controller indicates failure in the activity's execution, or the dispatcher detects that an activity's duration bound is violated, the method selection component is re-invoked. The control sequencer then updates its knowledge of any new constraints and selects an alternative set of methods that safely complete the RMPL program.

Self-Optimizing Methods through Safe, Decision-Theoretic Dispatch. In addition to failure, component performance can degrade dramatically, reducing system performance to unacceptable levels. To maintain optimal performance, predictive method dispatch utilizes decision-theoretic method dispatch, which continuously monitors performance and selects the currently optimal available set of methods that achieve each requisite function.

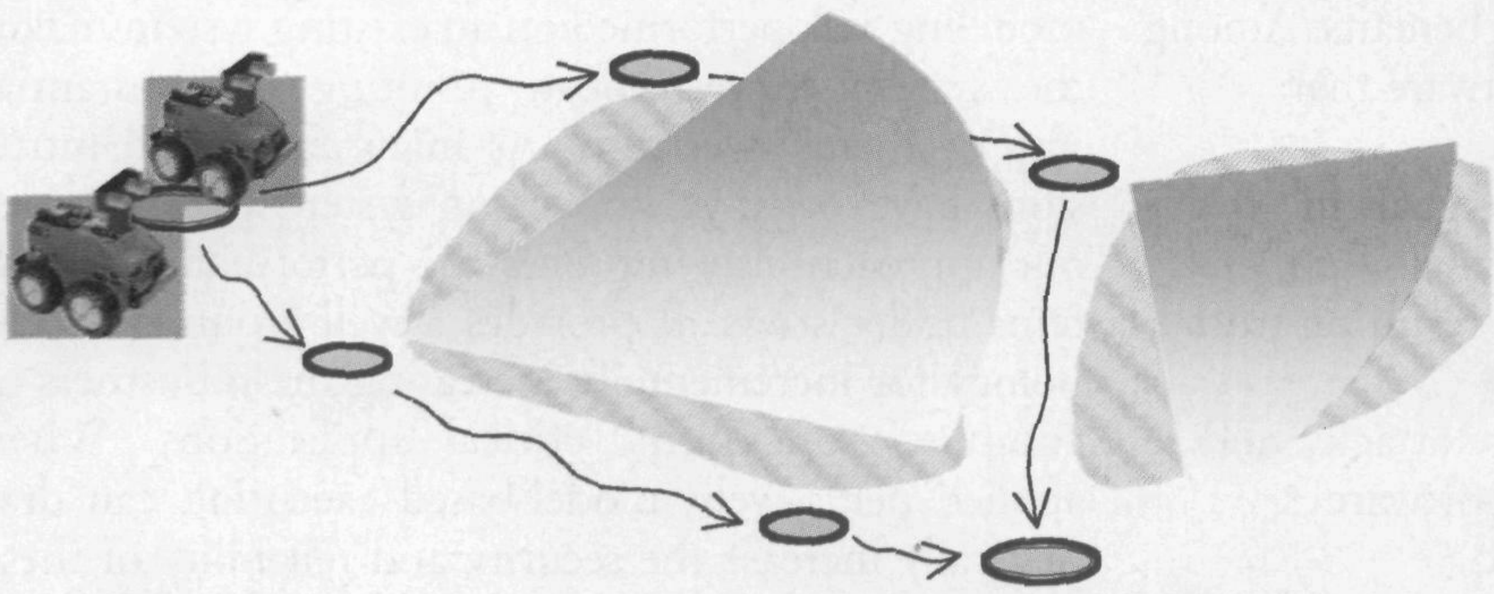

Figure 1. Rover testbed experimental platform.

Results

Initial testing of the described system has been performed by augmenting the MIT Model-Based Embedded and Robotic Systems rover testbed. The rover testbed consists of a fleet of all-terrain robot vehicles within a simulated Martian terrain. By way of example, we describe one mission whose robustness has been enhanced by the system.

Two rovers must cooperatively search for science targets in the simulated Martian terrain. This is done by having the rovers go to the selected vantage points looking for targets of interest using the rover's stereoscopic cameras. The rovers divide up the space so they can minimize the time taken in mapping the available science targets in the area. The paths of the rovers are planned in advance, given existing terrain maps. The plan runs without fail. Between them, the rovers successfully find all of the science targets that we have placed for them to find. The scenario is shown in Figure 1.

In the test scenario, two faults are introduced by placing a large rock that blocks Rover1's view of one of the designated areas. When Rover1 reaches its initial position to look for science targets, its cameras detect the unexpected rock obscuring its view. This