Immobile Robots: AI in the New Millennium*

Brian C. Williams and P. Pandurang Nayak
NASA Ames Research Center, MS 269-2
Recom Technologies
Moffett Field, CA 94305 USA
E-mail: {williams, nayak}@ptolemy.arc.nasa.gov

Abstract

A new generation of sensor rich, massively distributed, autonomous systems are being developed that have the potential for profound social, environmental, and economic change. These include networked building energy systems, autonomous space probes, chemical plant control systems, satellite constellations for remote ecosystem monitoring, power grids, biosphere-like life support systems, and reconfigurable traffic systems, to highlight but a few. To achieve high performance, these *immobile robots* (or *immobots*) will need to develop sophisticated regulatory and immune systems that accurately and robustly control their complex internal functions. To accomplish this, immobots will exploit a vast nervous system of sensors to model themselves and their environment on a grand scale. They will use these models to dramatically reconfigure themselves in order to survive decades of autonomous operations. Achieving these large scale modeling and configuration tasks will require a tight coupling between the higher level coordination function provided by symbolic reasoning, and the lower level autonomic processes of adaptive estimation and control. To be economically viable they will need to be programmable purely through high level compositional models. Self modeling and self configuration, coordinating autonomic functions through symbolic reasoning, and compositional, model-based programming are the three key elements of a *model-based autonomous systems* architecture that is taking us into the *New Millennium*.

Introduction

The lime-light has shifted dramatically in the last few years from an AI tradition of developing mobile robots, to that of software agents (aka softbots) (Etzioni & Segal 1992; Dent *et al.* 1992; Maes & Kozierok 1993; Kautz *et al.* 1994; Etzioni & Weld 1994). This is reflected in Hollywood, which has replaced R2D2 with cyberpunk descendants of Disney's Tron as the popular robot icon. One motivation for the shift is the difficulty and cost of developing and maintaining physical robots, which are believed to substantially impede progress towards AI's central goals of developing agent architectures and a theory of machine intelligence (Etzioni & Segal 1992). As Etzioni and Segal argue, software environments, such as a UNIX shell and the World-Wide Web, provide softbots with a set of ready-made sensors (e.g., **ls** and **gopher**) and end effectors (e.g., **ftp** and **telnet**) that are easy to maintain, while still providing a testbed for exploring issues of mobility and real-time constraints. At the same time the recent Internet gold rush and the ensuing Web literacy has provided an enormous textual corpus which screams for intelligent information gathering aides (Knoblock & Levy 1995; Levy, Rajaraman, & Ordille 1996).

Yet, two concerns have been raised about using software agents as a research testbed and application domain. First, softbots often operate in an environment lacking the rich constraints that stem from noisy, analog sensors and complex non-linear effectors, so fundamental to physical environments. Can such a software environment adequately drive research on agent kernels? Second, given that much of the information on the Internet is textual, will oft-envisioned softbot applications, such as information gathering and synthesis, be viable before the hard nut of language understanding has been cracked?

In this article we argue that the information gathering capabilities of the Internet, corporate intranets, and smaller networked computational systems supply additional testbeds for autonomous agents of a very different sort. These testbeds, which we call *immobile robots* (or *immobots*), have the richness that comes from interacting with physical environments, yet promise the ready availability associated with the networked software environment of softbots. Potential immobile robots include networked building energy systems, autonomous space probes, chemical plant control systems, satellite constellations for ecosystem monitoring, power grids, biosphere life support systems, and reconfigurable traffic systems. Conversion of these and other real time systems to immobile robots will be a

*This article is based on an invited talk given at the Third International Conference on AI Planning Systems(Williams 1996) driving force for profound social, environmental, and economic change. The power of a term like robot, mobot, or softbot is the way in which it sets free our imagination. The purpose of this article is to see where the immobot metaphor can take us, from science fiction, to fundamental AI insights, to significant applications.

We argue that the focus of attention of immobile robots is directed inward, toward maintaining their internal structure, as opposed to traditional robots, whose focus is on exploring and manipulating their external environment. This inward direction focuses the immobot on the control of its complex internal functions such as sensor monitoring and goal tracking, parameter estimation and learning, failure detection and isolation, fault diagnosis and avoidance, recovery and safing. Metaphorically speaking, the main functions of an immobot correspond to the human nervous, regulatory, and immune systems, rather than the navigation and perceptual systems being mimicked in mobile robots.

Finally, we argue that these immobots give rise to a new family of autonomous agent architectures, called *model-based autonomous systems*. Three properties of such systems are central. First, to achieve high performance, immobots will need to exploit a vast nervous system of sensors to model themselves and their environment on a grand scale. They will use these models to dramatically reconfigure themselves in order to survive decades of autonomous operations. Hence, self-modeling and self-configuration comprise an essential executive function of an immobot architecture. Second, to achieve these large scale modeling and configuration functions, an immobot architecture will require a tight coupling between the higher level coordination function provided by symbolic reasoning, and the lower level autonomic processes of adaptive estimation and control. Third, to be economically viable immobots will have to be programmable purely from high level compositional models, supporting a "plug and play" approach to software and hardware development.

The above properties of a model-based autonomous system are embodied in two implemented systems, *Moriarty* (Williams & Millar 1996) and *Livingstone* (Williams & Nayak 1996). Livingstone provides a kernel for controlling the immune system of immobile robots, while Moriarty provides part of the kernel for controlling an immobot's regulatory system. Our work on these systems fuses together research from such diverse areas of AI as model-based reasoning, qualitative reasoning, planning and scheduling, execution, propositional satisfiability, concurrent reactive languages, Markov processes, model-based learning, and adaptive systems. Moriarty and Livingstone are grounded in two immobot testbeds. Moriarty was part of the *Responsive Environment* (Zhang, Williams, & Elrod 1993; Elrod *et al.* 1993), an intelligent building control system developed within the Ubiquitous Computing project at Xerox PARC. Livingstone is part of the *Remote Agent*, a goal-directed, fully autonomous control architecture, which will fly NASA's Deep Space One space probe in 1998 (Pell *et al.* 1996a). At least one of these immobots has the promise of significant impact at the very beginning of the New Millennium.

Examples of Immobile Robots

Hardware advances in cheap single chip control and communication processors, sensors, point actuators, analog to digital converters, and networking has enabled a new category of autonomous system that is sensor rich, massively distributed, and largely immobile. This technology is being quickly embedded in almost every form of real time system, from networked building energy systems to spacecraft constellations. The current generation of these hybrid hardware/software systems has created a slumbering giant whose potential has only begun to be tapped. Furthermore they offer a diverse set of ready-made immobile robot testbeds, just waiting to be exploited by AI researchers.

The potential of this technology captured the imagination of Hollywood in the 60's, with such movies as *2001: A Space Odyssey, The Forbin Project, The Andromeda Strain*, and the *Star Trek* episode, *Spock's Brain*. The most famous computational intelligence, the HAL9000 computer, and the most famous biological intelligence, Spock, were both for a time immobile robots. In this article we use HAL and Spock as starting points for two immobile robot futures.

HAL9000

In the movie *2001,* HAL is the first of a new generation of computers with unprecedented intelligence and flawless behavior. HAL is characterized as the single mission element with the greatest responsibility, the "brain and central nervous system" of a spacecraft targeted for Jupiter.

It may seem puzzling to call HAL an immobot; after all he travels from Earth to Jupiter in 18 months, making him the fastest man made artifact of his time. However, traditional mobile robots typically focus on their external environment and on the tasks of navigation and obstacle avoidance. The movie, on the other hand, portrays HAL with an extreme sense of immobility. The camera zooms in for long periods on HAL's optical sensors, which are attached to walls throughout the spaceship. HAL's actuators are also extremely limited; examples include the opening and closing of doors, and the raising and lowering of beds. The distribution of these sensors and actuators throughout the spacecraft compensates for their immobility, giving HAL a sense of omnipresence. HAL's attention is focused inward throughout the movie. This is highlighted by the characterization of HAL as the "brain and central nervous system" of the spacecraft, as opposed to the navigator and pilot. HAL's major responsibility is to look after the health of the spacecraft and crew, including monitoring the spacecraft's health, performing fault diagnosis and repair, operating the life support systems, and continuously monitoring the medical status of crew members in hibernation. Finally, the movie highlights the connection between HAL's higher level symbolic reasoning and the complex, low level, autonomic processes distributed throughout the spacecraft. Hence HAL can be thought of as the spacecraft's *immune system.*

The New Millennium program

While many aspects of *2001* now seem farfetched, the current future of space exploration is no less exciting. With the creation of the New Millennium program in 1995, NASA has put forth the challenge of establishing a "virtual presence" in space through an armada of intelligent space probes that autonomously explore the nooks and crannies of the solar system:

> With autonomy we declare that no sphere is off limits. We will send our spacecraft to search beyond the horizon, accepting that we cannot directly control them, and relying on them to tell the tale. – *Bob Rasmussen, Cognizant Engineer, Cassini Mission.*

This "presence" is to be established at an Apollo-era pace, with software for the first probe to be delivered in mid-1997, leaving only a year and a half for development, and the probe (Deep Space One) to be launched in mid-1998. The additional constraint of low cost is of an equal magnitude. Unlike the billion dollar Galileo and Cassini missions with hundreds of members in their flight operations team, New Millennium missions are to cost under $100 million with only tens of flight operators. The eventual goal is a $50 million spacecraft operated by a mere handful of people. Achieving these goals will require autonomy and robustness on an unprecedented scale.

The final challenge, spacecraft complexity, is equally daunting. Consider Cassini, NASA's state of the art spacecraft headed for Saturn. Cassini's "nervous system" includes a sophisticated networked, multi-processor system, consisting of two flight computers that communicate over a bus to more than two dozen control units and drivers. These establish complex sensing and control paths to an array of fixed sensors and actuators, including inertial reference units, sun sensors, pressure sensors, thrusters, main engines, reaction wheels and heaters. The most complex is the main engine subsystem, consisting of a web of redundant pipes, valves, tanks, and engines that offer exceptional levels of reconfigurability. Coordinating these complicated, hybrid systems poses significant technical hurdles.

Together, the goals of the New Millennium program pose an extraordinary opportunity and challenge for AI. As with HAL, our challenge is to provide the immune system for this type of immobile robot over the lifetime of its mission.

Spock's Brain

A second example of an immobile robot from 60's Hollywood comes from a little known episode of the original Star Trek series, called "Spock's Brain." In this episode Spock's body is found robbed of its brain by an unknown alien race, and the crew of the Enterprise embarks upon a search of the galaxy in order to reunite Spock's brain and body. Spock detects that he has a new body that stretches into infinity, and which appears to be breathing, pumping blood and maintaining physiological temperature. When discovered, Spock's brain is found to be within a black box, tied in by light rays to a complex control panel. Instead of breathing, maintaining temperature and pumping blood, he is recirculating air, running heating plants and recirculating water. That is, the function that requires this supreme intelligence is the regulation of a planet-wide heating and ventilation system.

As with HAL, Spock's body in this case is extremely immobile. The episode portrays an immobile robot as a massively distributed behemoth, sufficient to encircle a globe. Again the crucial link between high level reasoning (i.e., Spock's brain) and autonomic processes is highlighted. Finally, while *2001* highlights HAL's function as an immune system that maintains the health of the immobot, this episode highlights Spock's function as a regulatory system, performing high fidelity control of the immobot's internal organs. A regulatory system provides an efficient metabolism, supplying central resources efficiently, equitably distributing these resources throughout the system, consuming these resources efficiently, and attenuating the effects of disturbances.

The Responsive Environment

The concept in *Spock's Brain* is not as implausible as it might seem. In fact it is quickly becoming a reality through a confluence of forces (Figure 1). First is the broad installation of the new generation of networked building management systems. These are networked multi-processor systems containing hundreds to thousands of processors that allow high fidelity sensing and control throughout a building. They will enable systems that are far more energy efficient, and at the same time improve indoor air quality. Second is the interconnection of building and home utilities through optical fiber networks. This has lead to several commercial alliances between members of the utility and telecommunication industries. Third is the deregulation of the utilities, and the ensuing establishment of computer-based energy markets. These allow peak and average electrical rates to be fluidly adjusted, enabling more even and stable balancing of the energy load. Technologies for storing energy within buildings during off peak hours will enable even greater stability and efficiency. The rapid deployment of these networked control technologies has already generated a multibillion dollar growth in the service industries for remote build-

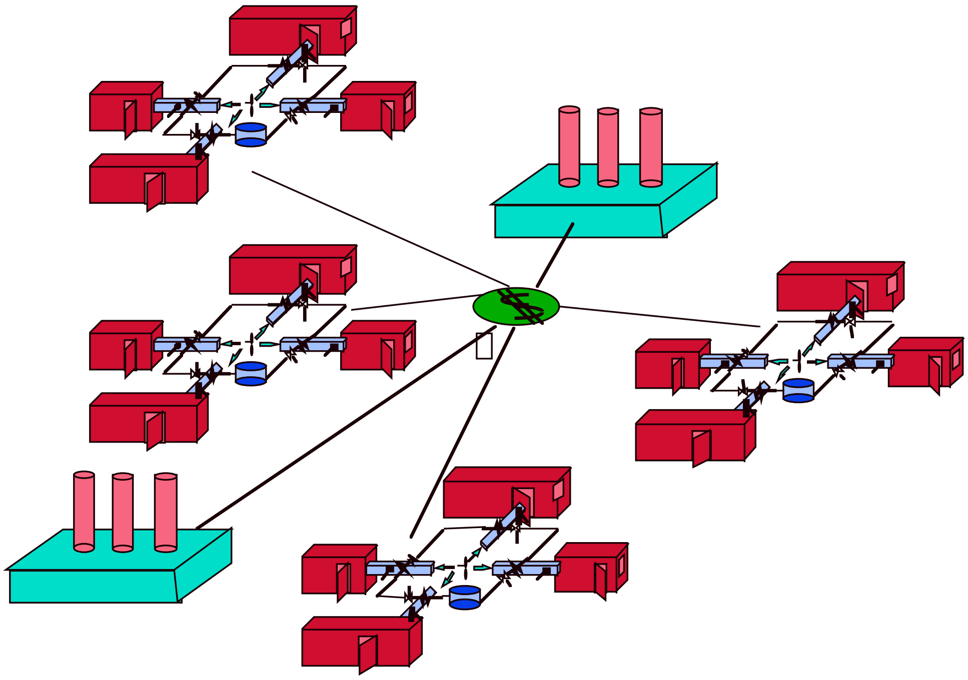

Figure 1: Immobots of unprecedented size are being developed in the area of building energy management. Key elements include networked building control systems, optical networks that connect buildings on a power grid to multiple power production facilities, deregulatation of the power industry, and the establishment of a dynamic energy market.

ing monitoring and maintenance. But the potential of these immobile robots for optimal energy efficient control has yet to be tapped.

These are only two of an ever increasing collection of immobile robot testbeds being constructed. NASA's Earth observing system is moving towards a level of sensing that will enable full Earth ecosystem modeling, providing insight into problems of pollution, global warming, and ozone depletion. Vast networked control systems are generating a revolution in chemical processing, factory automation, drug manufacturing, and semiconductor fabrication, producing immobile robots that enable substantial improvements in quality, efficiency and safety.

These are not far off dreams. Many government and private institutions, such as NASA, PG&E, Rockwell Automation, Echelon, Hewlett Packard and Johnson Controls, are aggressively embracing these technologies as fundamental to their future in the new millennium, that is, in the immediate future. The two immobot testbeds discussed in this article represent first steps towards these future visions.

Immobot Characteristics

From these examples we can extract a number of properties that distinguish immobots from their mobile robot and softbot siblings, both in terms of their physical structure and their most salient functions. Structurally, immobots are massively distributed, physically embedded, autonomous systems with a large array of simple, fixed location sensors and actuators. Functionally an immobot's primary task is to control its massive regulatory, immune, and nervous systems through a coupling of high-level reasoning, to adaptive autonomic processes. More specifically, an immobot has the following distinctive features:

Physically Embedded An immobot's sensors and actuators operate on the physical world, many of which operate in noisy environments. Immobots need to react in real-time; however, the timescale of the reaction time can vary dramatically. For example, a spacecraft may have to act instantaneously to close off a leaking thruster, but then may have weeks during its cruise to perform a self diagnosis.

Immobile An immobot's sensors and actuators reside at fixed locations, and are largely limited to one dimensional signals and one degree of freedom movement. Examples include light sensors, window blind actuators, gyroscopes, valves, and reaction wheels. While an immobot's hardware is often redundant and highly reconfigurable, the primitive elements of interaction that are used to build a configuration are simple and fixed. In contrast, a traditional robot typically has a set of complex mobile sensors and actuators, such as 3D vision systems and articulated hands, arms, and legs.

Omnipresent The lack of sensor and actuator mobility is compensated for by their sheer number. A spacecraft has dozens of sensors and actuators, a building can have on the order of thousands. Sensors and actuators must reside *a priori* in all locations that the immobot wants to model or affect, giving the immobot the potential for being continuously aware of all aspects of its environment. In contrast, a mobile robot's few precious sensors and actuators must be navigated to the point where they are needed. The omnipresence of the immobot shifts the bottleneck from the expense of gathering each piece of information, to the integration of a continuous stream of data.

Massively Distributed & Tightly Coupled Computation is distributed along lengthy communication paths, with increasingly complex computation being performed at sites near sensors and actuators. For example, in Cassini, a complex set of physical and software processes work together in order to fire an engine – the flight computer turns on an engine by sending a command over a bus to a communication processor, which tells a driver to open two valves, which causes fuel to flow into the engine, where combustion occurs. These properties lead to distributed sensing and control problems of high dimensionality due to tight couplings between internal components. This high dimensionality makes these problems extremely difficult to solve.

Self-Absorbed Mobile robots and softbots focus largely on what is occurring in the world outside the "bot," including navigating around obstacles, moving through networks to databases, and changing navigation paths due to external failures. On the other hand, an immobile robot's attention is largely directed inward toward monitoring the internal health of its network (immunology) and reconfiguring its components or control policies to achieve robust performance (regulation). While some reasoning is directed outward, the external world is not fluid in the same sense as for classical robots.

One of a Kind Ironically what binds together immobile robots is that no two are alike. Drug manufacturing lines, Disney's space mountain, car factories, Mars rovers, Earth orbiting satellites, deep space probes, and Antarctic biospheres, are each one of a kind devices, making it difficult to amortize their development costs. The dilemma is how to cost effectively build these one of a kinds, while providing high performance and reliability.

Controlling Immobots

In the preceding sections we argued that the dominant function of an immobot is to control its massive regulatory, immune, and nervous systems through a coupling of high-level reasoning with adaptive autonomic processes. Providing a regulatory and immune system involves a broad set of tasks, including those listed in Figure 2.

- parameter estimation
- monitoring
- mode confirmation
- goal tracking
- detecting anomalies
- isolating faults
- diagnosing causes
- calibration
- adaptive control
- control policy coordination
- hardware reconfiguration
- fault recovery
- standby
- safing
- fault avoidance

Figure 2: Tasks performed to maintain the immobots regulatory and immune systems.

Writing software to control the immune and regulatory systems of an immobot is difficult for a variety of reasons. First, it often requires the programmer to reason through system-wide interactions to perform the appropriate function. For example, diagnosing a failed thruster requires reasoning about the interactions between the thrusters, the attitude controller, the star tracker, the bus controller, and the thruster valve electronics. The complexity of these interactions can lead to cognitive overload, causing suboptimal decisions and even outright errors. Furthermore, the one of a kind nature of immobots means that the cost of reasoning through system-wide interactions cannot be amortized, and must be paid over again for each new immobot.

Second, the need for fast reactions in anomalous situations has led to a tradition of precomputing all responses. However, immobots often operate in harsh environments over decades, so that explicitly enumerating responses to all possible situations quickly becomes intractable. Tractability is usually restored with the use of simplifying assumptions such as using local suboptimal control laws, assuming single faults, ignoring sensor information, or ignoring subsystem interactions. Unfortunately, this results in systems that are either brittle or grossly inefficient.

Finally, the autonomic processes of an immobot involve a broad range of discrete, continuous, and software behaviors, including control laws, Kalman filters, digital hardware commands, and software drivers. This range of behaviors needed by an immobot makes it very difficult and expensive to both manually synthesize high fidelity autonomic processes and couple these autonomic processes to high-level symbolic reasoning.

Model-based Autonomous Systems

The goal of our research program is to solve the above difficulties by developing a *model-based autonomous system kernel* for maintaining the regulatory and immune systems of immobots. The kernel that we are driving towards is defined by three desiderata: *model-based programming*, *model-based reactive execution*, and *model-based hybrid systems*.

Our work on immobots originated with work on interaction-based design (Williams 1990), which explored the coordination and construction of continuous interactions, as a design task that involves the addition of novel hardware device topologies. In contrast model-based autonomy explores the coordination of hardware and software interactions using a digital controller.

Model-based Programming

A model-based autonomous system addresses the difficulty of reasoning about system-wide interactions using model-based programming. Model-based programming is based on the idea that the most effective way to amortize software development cost is to make the software plug and play. To support plug and play, immobots are programmed by specifying component models of hardware and software behaviors. A model-based autonomous system combines component models to automate all reasoning about system wide interactions necessary to synthesize real-time behaviors like the ones in Figure 2. The development of model libraries is used to reduce design time, facilitate reuse, and amortize modeling costs.

The application of a stringent qualitative modeling methodology is used to reduce both modeling time and the sensitivity to model inaccuracies and hardware changes. Our experience and those of others (Hamscher 1991; Malik & Struss 1996) has shown that extremely weak qualitative representations, e.g., representing only deviations from nominal behavior, are quite sufficient for many model-based autonomy tasks. Counter to the folklore of the field (Sacks & Doyle 1992), ambiguity and intractable branching has not proven to be a significant practical issue.

Model-based programming on a large-scale is supported by developing languages, compilers, debuggers, and visualization tools that incorporate classical concepts of object-oriented, procedural, and hierarchical abstractions into the modeling language.

Model-based Reactive Execution

The difficulty of precomputing all responses means that a model-based autonomous system must use its models to synthesize timely responses to anomalous and unexpected situations at execution time. Furthermore, the need to respond correctly in time critical and novel situations means that it must perform *deliberative reasoning about the model within the reactive control loop*. While the list of tasks that the model-based execution system must support is seemingly diverse (Figure 2), they divide into two basic functions: self modeling and self configuration.

Self Modeling Identifying anomalous and unexpected situations, and synthesizing correct responses in a timely manner requires an immobot to be self-modeling. While parts of its model are provided *a priori* using model-based programming, other parts need to be adapted or elaborated using sensor information. Self modeling tasks include tracking model parameters over time (e.g., base line voltages), tracking changes in component behavioral modes (e.g., a valve going from open to closed, and then to stuck closed), and elaborating quantitative details of qualitative models (e.g., learning a quantitative pipe model, given that flow is proportional to the pressure drop). Self modeling tasks are listed in the left hand column of Figure 2; all but estimation, monitoring and calibration involve identifying discrete behavioral modes of the immobot's components.

Self Configuration To provide immune and regulatory systems, an immobot must be self-configuring; it must dynamically engage and disengage component operating modes and adaptive control policies in response to changes in goals, the immobot's internal structure, and the external environment. The right hand column of Figure 2 lists tasks that involve self-configuration. The first two items involve coordinating or adjusting continuous control policies, while the remainder involve changes in component behavioral modes.

Model-based Hybrid Systems

Given the wide range of digital, analog, and software behaviors exhibited by an immobot, developing a model-based approach for coordinating the immobot's autonomic processes requires a rich modeling language and reasoning methods that go well beyond those traditionally used in qualitative and model-based diagnosis. More specifically, a model-based autonomous system must be able to represent and reason about:

Concurrent software Coordinating, invoking, and monitoring real-time software requires formal specifications of their behavior. These are modeled by incorporating formal specifications of concurrent transition systems into the model-based programming language. Concurrent transitions systems provides an adequate formal semantics for most concurrent real-time languages (Manna & Pnueli 1992).

Continuous adaptive processes Achieving high fidelity requires the merging of symbolic model-based methods with novel adaptive estimation and control techniques (e.g., neural nets). Specifications of adaptive processes must be combined with specifications of discrete concurrent behavior within the models. For high-level reasoning systems that coordinate adaptive processes, the most suitable specification of the adaptive processes are often qualitative.

Stochastic processes Inherent to supporting an immobot's regulatory and immune functions is the modeling and control of stochastic events that occur within an immobot's components. This stochastic behavior is modeled by extending the concurrent transition system model, mentioned above, to modeling concurrent, partially observable Markov processes.

The previous three subsections outlined the key requirements for model-based autonomy. The essential property that makes these requirements manageable is the relative immobility of our robots. The system interactions are relatively fixed and known *a priori* through the component models and their interconnection. The flexibility within the system is largely limited to changes in component modes and control policies, and adjustments to parameter values. In the rest of this article we consider two implemented systems that exploit immobility to achieve the above desiderata. They form two major components of a kernel we envision for maintaining the regulatory and immune systems of immobots.

Responsive Environments

The scenario in "Spock's brain" of a heating and cooling system on a planetary scale highlighted three aspects of immobile robots that we will examine technically in this section. First is the task of maintaining a massive, high performance regulatory system. Second is the need by this system to acquire and adapt accurate models of itself and its environment, in order to achieve high performance. Third is the need for high level reasoning to coordinate a vast set of autonomic processes during adaptive modeling and regulation.

As a stepping stone towards this ambitious scenario, we developed a testbed for fine grained sensing and control of a suite of offices at Xerox PARC, called the *responsive environment* (Elrod *et al.* 1993). This testbed includes a networked control system for the complete building, plus fifteen model offices each of which has been enhanced with a networked microprocessor that controls an array of sensors and actuators. These include sensors for occupancy, light-level, temperature, pressure and air flow, and actuators for regulating air temperature and air flow.

The heart of the building is the central plant, which generates air flow, and cold and hot water through a fan, chiller and boiler, respectively. The extremities of the building are the hundred or more offices, whose temperature must be carefully regulated. The veins and arteries of the building are the pipes and duct work that deliver air and water to the extremities, where they are used to regulate temperature. The connection between the central plant and a single office, used for cooling, is shown in Figure 3. Office temperature is controlled through heat flow in or out of the office. This includes heat flowing from the sun and equipment, through the walls and doorways, and via the air duct. Heat flow into an office is controlled via the air duct in two ways. The amount of air flow is controlled by a damper, that partially blocks airflow. The temperature of the air is changed by blowing the air over a radiator like device, called a reheat, that contains hot or cold water. The flow rate of the water is controlled by the reheat valve position.

What makes regulating a building difficult is the overwhelming number of control variables and the fact that the control variables are highly coupled. For example, the energy consumption of the fan and chiller are nonlinear functions of the fan speed and the change from outside to inside temperature. The optimal setting of the damper and reheat valve, then depend on the outside temperature and the demands that other offices are placing on the chiller and fan. Hence the performance of all the offices are coupled. This is exacerbated by the slow response time of office temperature change, which makes a trial and error approach to global control extremely unstable.

We solve this problem with a gradient descent controller that uses a global model to adaptively predict where the optima lies. An informal evaluation using the responsive environment testbed suggests that energy savings in excess of 30% are conceivable with help from model-based control (Zhang, Williams, & Elrod 1993).

Generic thermal models are available for complete buildings (e.g., the DOE2 simulator models) that suffice for this type of control. These models are extremely rich, including not only the building's hardware and nonlinear thermal characteristics, but sunlight and shading, external weather, and the occupants' desires. They have on the order of thousands of equations for buildings with one hundred or more offices.

The labor intensive task is tailoring this model to the specifics of a building, which requires estimating numerical values for parameters, like thermal conductance through the walls, heat output of the equipment, and thermal capacity of the office air space. An immobot can estimate its parameters by adjusting their values until the model best fits the sensor data. More precisely, given sensed variables y and $\mathbf{x}$ (a vector), and a vector of parameters $\mathbf{p}$, we first construct an estimator f from the model, that predicts y given $\mathbf{x}$ and $\mathbf{p}$:

$$y = f(\mathbf{x}; \mathbf{p})$$

Next, given a set of $(\mathbf{x}, y)$ data D, estimating $\mathbf{p}$ using least squares fit of f to y involves solving the optimization problem:

$$\mathbf{p}^* = arg \min_{\mathbf{p}} \sum_{\langle y_i, \mathbf{x}_i \rangle \in D} (y_i - f(\mathbf{x}_i; \mathbf{p}))^2$$

Such an estimate is performed by an adaptive numerical algorithm that can be viewed as one of the immobot's autonomic processes.

There are far too many parameters in a building model to estimate all of them at once. Instead the immobile robot must automate a control engineer or modeler's expertise at breaking a model into a set of

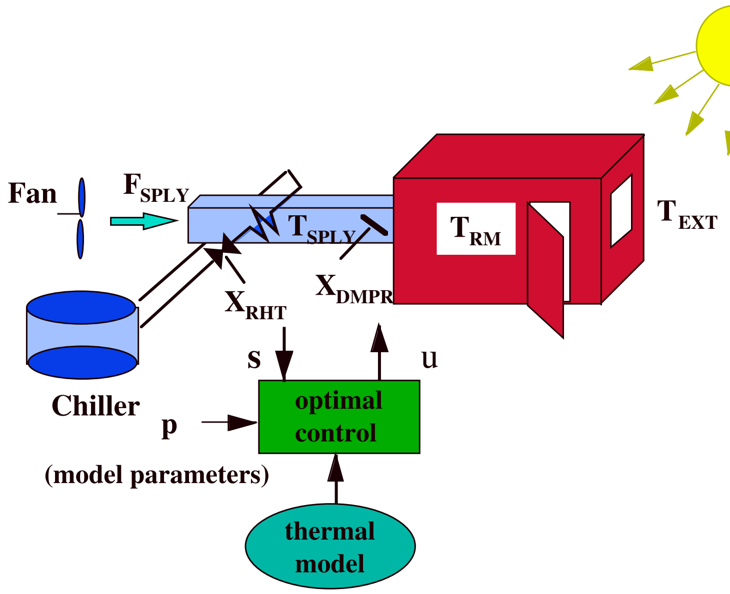

Figure 3: Model-based air conditioning of a single office, using air blown over pipes that contain a flow of chilled water.

tractable parameter estimation tasks[1], and then coordinating and combining their results. This coordination represents the link between high-level reasoning and autonomic processes in our self modeling immobot. An army of modelers is employed for large scale tasks, such as earth ecosystem modeling, at great cost. This army must be automated for many immobile robot tasks, if high performance and robustness is to be achieved. Moriarty automates key aspects of how a community of modelers decompose, simplify, plan and coordinate large scale model estimation tasks, through a technique called *decompositional, model-based learning (DML)*.

Coordinating Adaptive Model Estimation

When decomposing a model into a set of estimation tasks there is often a large set of possible estimators ($y = f(\mathbf{x}; \mathbf{p})$) to choose from, and the number of parameters contained in each estimator varies widely. Moriarty decomposes a model into a set of "simplest" estimators that minimize the dimensionality of the search space and the number of local minima, hence improving learning rate and accuracy. Each estimator together with the appropriate subset of sensor data forms a primitive estimation action. Moriarty then plans the ordering and coordination of information flow between them.

To estimate the parameters for a single office, Moriarty starts with a thermal model consisting of fourteen equations involving seventeen state variables and eleven parameters. About a third of the equations are nonlinear, such as:

$$F_{dmpr} = \left(\frac{\rho_{dmpr}(X_{dmpr})}{R_{dct}}\right)\sqrt{P_{dct}}$$

which relates air flow through the damper to duct pressure and duct air resistance as a function of damper position. Nine of the state variables are sensed, including temperature T, flow rate F, air pressure P, damper and reheat valve position X. Seven of the eleven parameter values are unknown and must be estimated by Moriarty.

Moriarty's task is to generate an estimation plan from the thermal model. Moriarty, when brought up to full capability will be able to generate the plan in Figure 4 (it currently generates less efficient plans). In terms of the immobot metaphor, each octagon in the figure represents an adaptive, autonomic estimation process, and the plan graph represents higher level coordination. Each octagon in the diagram is an *estimation action* that is defined by an estimator $y = f(\mathbf{x}, \mathbf{p})$, applied to a subset of the sensor data. The estimator is constructed from a subset of the model. The arc going into the action specifies additional parameters whose values are already known, and the arc leaving the action specifies parameters whose values have been determined by the estimation. For example, the top octagon, labeled "air flow", produces an estimate for parameter R_{dct}, the air resistance in the duct. It performs this estimate using the following estimator for F_{ext},

$$F_{ext} = (\rho_{lkg} + \rho_{dmpr}(X_{dmpr}))\,\frac{\sqrt{P_{dct}}}{R_{dct}}$$

[1] To achieve tractability a modeler of nonlinear physical systems will typically search for estimators containing four or less parameters.

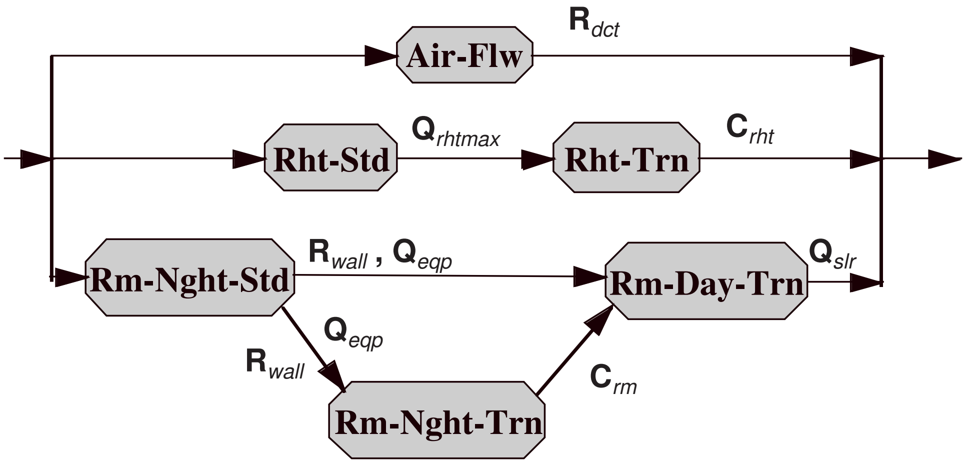

Figure 4: Model-estimation plan for a single office.

derived using three equations in the office model pertaining to airflow. This action does not take any parameter values as input (ρ_{lkg} and $\rho_{dmpr}(X_{dmpr})$ are known a priori as constants).

Consider a common sense account of how the estimation plan is generated. Moriarty constructs this plan by generating a set of possible estimation actions from the model. It then selects and sequences a subset of the actions sufficient to cover all parameters. Actions are generated in two steps. In the first step subsets of the model and sensors are identified that are sufficiently constrained to form an estimation problem. For example, the "air flow" action described above was generated from the pressure P_{dct} and air flow sensors F, together with the following air flow equations:

$$F_{ext} = F_{lkg} + F_{dmpr}$$

$$F_{lkg} = \left(\frac{\rho_{lkg}}{R_{dct}}\right)\sqrt{P_{dct}}$$

$$F_{dmpr} = \left(\frac{\rho_{dmpr}(X_{dmpr})}{R_{dct}}\right)\sqrt{P_{dct}}$$

Two additional actions, "Rht" and "Rm", are created from the set of temperature and heat flow equations, by using the duct air temperature sensor to split the set into reheat and room submodels. The first action, "Air Flw" involves the single parameter R_{dct}, action "Rht" involves the two parameters Q_{rhtmax} and C_{rht}, and "Rm" involves the four parameters, R_{wall}, Q_{eqp}, C_{rm}, and $Q_{slr}(t)$. Moriarty's first step generates eight possible estimation actions in total, each containing between one and seven parameters. "Air Flw", "Rht" and "Rm" are three actions that cover all seven model parameters and contain the fewest parameters individually. The fact that the sets of parameters in these actions are disjoint is purely coincidental.

An additional step, under development, produces one or more simplified versions of each estimation action by identifying conditions on the data such that influences by one or more parameters becomes negligible. For example, consider the estimator for action "Rm":

$$\frac{dT_{rm}}{dt} = \frac{C_0 F_{sply}\left(T_{sply} - T_{rm}\right)}{C_{rm}} + \frac{Q_{eqp} + Q_{slr}(t) + R_{wall}\left(T_{ext} - T_{rm}\right)}{C_{rm}}$$

This estimator can be simplified by noticing that solar effect $Q_{slr}(t)$ is negligible at night time, when the sun is down (action "Rm-Nght"), while it is significant during the day (action "Rm-Day-Trn"). Action "Rm-Nght" is generated from "Rm" by restricting the data set to data taken at night. This allows $Q_{slr}(t)$ to be eliminated, reducing the number of parameters in the estimator from four to the three parameters R_{wall}, Q_{eqp}, and C_{rm}.

Furthermore, the effect of the room heat capacity C_{rm} is negligible when the room temperature is in steady state ("Rm-Nght-Std"), while it becomes particularly significant during room temperature transients ("Rm-Nght-Trn"). Hence action "Rm-Nght-Std" is generated from the simplified action "Rm-Nght", by further restricting the data set to data where $\frac{dT_{rm}}{dt}$ is small. This allows the term $\frac{dT_{rm}}{dt}C_{rm}$ to be eliminated from the estimator and reduces the number of parameters from three to two. Applying this simplification process to the first three of the eight actions generated in the first step results in the six primitive estimation actions shown in Figure 4.

Having generated these estimation actions, sequencing exploits the fact that some of these estimation actions share parameters, in order to further reduce the dimensionality of the search performed by each action. In particular, the output of an estimation action with fewer unknown parameters, such as "Rm-Nght-Std", is passed to an overlapping action with more parameters, such as "Rm-Nght". This reduces the number of unknown parameters estimated in the second action; for example, "Rm-Nght" is left only with C_{rm} as an unknown. This approach produces the sequence shown in Figure 4. Each action estimates at most two unknown parameters, a dramatic reduction from the seven unknown parameters in the original problem.

Technically, in the first step Moriarty decomposes a model into an initial set of estimation actions by exploiting an analogy to the way in which model-based diagnosis decomposes and solves large scale multiple fault problems. The decomposition of a diagnostic problem is based on the concept of a *conflict*—a minimal subset of a model (typically in propositional or first order logic) that is inconsistent with the set of observations (de Kleer & Williams 1987; Reiter 1987). Moriarty's decompositional learning method is based on the analogous concept of a *dissent*—a minimal subset of an algebraic model that is *overdetermined* given a set of sensed variables (*i. e.,* a dissent is just sufficient to induce an error function). The set of three flow equations used earlier to construct the "Air flw" estimator is an example of a dissent. Following this analogy Moriarty uses a dissent generation algorithm (Williams & Millar 1996) that parallels the conflict recognition phase of model-based diagnosis.

Moriarty's simplification step (currently under development), is based on an order of magnitude simplification method, called *caricatural modeling,* (Williams & Raiman 1994). Caricatural modeling partitions a model into a set of operating regions in which some behaviors dominate and others become negligible. Moriarty applies caricatures to each estimation action generated in the first step, using the action's estimator as the model. Each operating region generated corresponds to a simplified estimation action; the operating region's model is the simplified estimator, and the operating region's boundary description provides a filter on data points for that estimator. In the thermal example, one of many simplifications to our estimator for "Rm" is to minimize $Q_{slr}(t)$. Since $Q_{slr}(t) \approx 0$ for all data at night, this simplification can be used. In the final step, sequencing selects and orders primitive estimation actions using an algorithm that greedily minimizes the unknown parameters in each estimator, as described in (Williams & Millar 1996).

Consider Moriarty's performance on the example, where the simplification step hasn't been performed (Figure 5). Moriarty generates the eight estimators, F1–F8, in the first step, and immediately sequences them to produce $\langle F2, F1, F6\rangle$, which corresponds to the flow reheat and room estimation actions, given earlier. We compare the performance of the eight estimators by running them against sensor data sets ranging in size from 10 to 200, shown above. The y axis denotes time required to converge on a final estimate of parameters involved in the dissent. The plot labeled F7 is for the original seven dimensional estimator, while plots F2, F1, and F6 are the three estimators in Moriarty's sequence. Higher dimensional estimators, like F7, tend to fail to converge given arbitrary initial conditions, hence ball-park initial parameter estimates were supplied to allow convergence in the higher dimensional cases. Decomposition leads to significant speed-up even when good initial estimates were available. For example, at trial size 200, the original estimator requires 166 seconds, while the total time to estimate all parameters using F2, F1, and F6 is under 9 seconds. This represents a speed up by a factor of 14.

In addition the sequence requires less data to converge. This is important for self-modeling immobile robots, which use online estimation to quickly track time-varying parameters. Employing the rule of thumb that the data set size should be roughly ten fold the dimension of the parameter space, F7 would require around 70 data points, while the F2, F1, F6 sequence requires only 40. The anomalous slow down in the convergence rate of F7 at 25 data points is attributed to insufficient data. Finally, although not included, parameter accuracy, measured by the confidence interval of each parameter is also improved using the generated sequence.

To summarize, a model-based approach is essential for regulating systems of the size of most immobile robots. Embodying an immobile robot with self-modeling capabilities requires the use of symbolic reasoning to coordinate a large set of autonomic estimation processes. This coordination mimics the way in which a community of modelers decomposes, simplifies and coordinates modeling problems on a grand challenge scale. DML automates one aspect of this rich model decomposition and analysis planning process.

The New Millennium Program

HAL in *2001* highlights four aspects of the immune system of an immobile robot that we examine technically in this section. First is the ability to monitor internal health, detecting and isolating failing functions, processes and components. Second is the ability to continually reconfigure the low level autonomic processes of an immobot, establishing intended functions and working around failures in a cost efficient and reliable manner. Third is the ability to reason extensively through reconfiguration options while ensuring reactivity. Fourth is the ability to reason about hybrid systems.

Livingstone

We have developed a system called *Livingstone*[2] to investigate these issues in the context of NASA's New Millennium program. Livingstone is a fast, reactive,

[2] Livingstone, the program, is named after David Livingstone (1813-1873), the 19th century medical missionary and explorer. Like David Livingstone, Livingstone the program is concerned with exploration and the health of explorers.

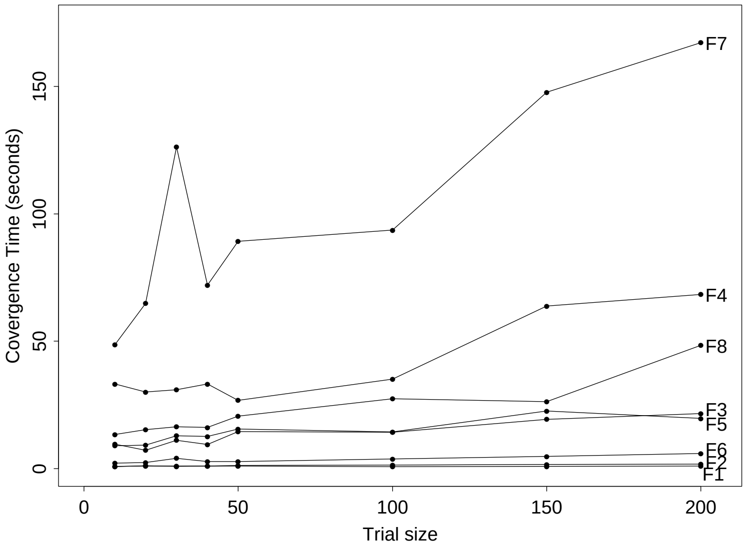

Figure 5: Convergence rate vs data size for the generated estimators F1–F8.

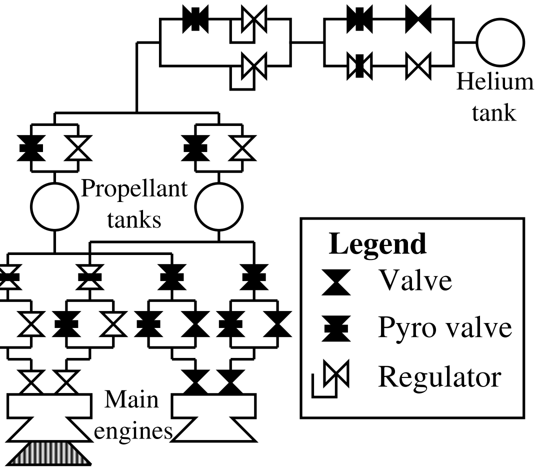

Figure 6: Schematic of the Cassini main engine subsystem. In the valve configuration shown, the engine on the left is firing.

model-based configuration manager. Using a hierarchical control metaphor, Livingstone sits at the nexus between the high-level feedforward reasoning of classical planning/scheduling systems, and the low-level feedback response of continuous adaptive control methods, providing a kernel for model-based autonomy. Livingstone is distinguished from more traditional robotic executives through the use of deliberative reasoning in the reactive feedback loop. This deliberative reasoning is compositional, model-based, can entertain an enormous search space of feasible solutions, and yet is extremely efficient due to the ability to quickly focus on the few solutions that are near optimal.

Three technical features of Livingstone are particularly worth highlighting. First, the approach unifies the dichotomy within AI between deduction and reactivity (Agre & Chapman 1987; Brooks 1991). We achieve a reactive system that performs significant deduction in the sense/response loop by drawing on our past experience at building fast propositional conflict-based algorithms for model-based diagnosis, and by framing a model-based configuration manager as a propositional feedback controller that generates focused, optimal responses. Second, Livingstone's representation formalism achieves broad coverage of hybrid discrete/continuous, software/hardware systems by coupling the concurrent transition system models underlying concurrent reactive languages (Manna & Pnueli 1992) with the qualitative representations developed in model-based reasoning. Third, the long held vision of model-based reasoning has been to use a single central model to support a diversity of engineering tasks. For model-based autonomous systems this means using a single model to support a variety of execution tasks including tracking planner goals, confirming hardware modes, reconfiguring hardware, detecting anomalies, isolating faults, diagnosis, fault recovery, and safing. Livingstone automates all these tasks using a single model and a single core algorithm, thus making significant progress towards achieving the model-based vision.

Configuration management

To understand the role of a configuration manager, consider Figure 6. It shows a simplified schematic of the main engine subsystem of Cassini, the most complex spacecraft built to date. It consists of a helium tank, a fuel tank, an oxidizer tank, a pair of main engines, regulators, latch valves, pyro valves, and pipes. The helium tank pressurizes the two propellant tanks, with the regulators acting to reduce the high helium pressure to a lower working pressure. When propellant paths to a main engine are open, the pressurized tanks force fuel and oxidizer into the main engine, where they combine and spontaneously ignite, producing thrust. The pyro valves can be fired exactly once, i.e., they can change state exactly once, either from open to closed or vice versa. Their function is to isolate parts of the main engine subsystem until needed, or to isolate failed parts. The latch valves are controlled using valve drivers (not shown), and an accelerometer (not shown) senses the thrust generated by the main engines.

Starting from the configuration shown in the figure, the high-level goal of producing thrust can be achieved using a variety of different configurations: thrust can be provided by either main engine, and there are a number of different ways of opening propellant paths to either main engine. For example, thrust can be provided by opening the latch valves leading to the engine on the left, or by firing a pair of pyros and opening a set of latch valves leading to the engine on the right. Other configurations correspond to various combinations of pyro firings. The different configurations have different characteristics since pyro firings are irreversible actions and since firing pyro valves requires significantly more power than opening or closing latch valves.

Suppose that the main engine subsystem has been configured to provide thrust from the left main engine by opening the latch valves leading to it. Suppose that this engine fails, e.g., by overheating, so that it fails to provide the desired thrust. To ensure that the desired thrust is provided even in this situation, the spacecraft must be transitioned to a new configuration in which thrust is now provided by the main engine on the right. Ideally, this is achieved by firing the two pyro valves leading to the right side, and opening the remaining latch valves (rather than firing additional pyro valves).

A configuration manager constantly attempts to move the spacecraft into lowest cost configurations that achieve a set of high-level dynamically changing goals, such as the goal of providing nominal thrust. When the spacecraft strays from the chosen configuration due to failures, the configuration manager analyzes sensor data to identify the current configuration of the spacecraft, and then moves the spacecraft to a new configuration which, once again, achieves the desired configuration goals. In this sense a configuration manager like Livingstone is a discrete control system that is strategically situated between high-level planning and low-level control; it ensures that the spacecraft's configuration always achieves the set point defined by the configuration goals.

Model-based configuration management

Livingstone is a reactive configuration manager that uses a compositional, component-based model of the spacecraft to determine configuration actions (see Figure 7). Each component is modeled as a transition system that specifies the behaviors of operating and failure modes of the component, nominal and failure transitions between modes, and the costs and likelihoods of transitions (see Figure 8). Mode behaviors are specified using formulas in propositional logic, while transitions between modes are specified using formulas in a restricted temporal, propositional logic. The restricted propositional temporal logic is adequate for modeling digital hardware, analog hardware using qualitative abstractions (Weld & de Kleer 1990; de Kleer & Williams 1991), and real-time software using the models of concurrent reactive systems in (Manna & Pnueli 1992). The spacecraft transition system model is a composition of its component transition systems in which the set of configurations of the spacecraft is the cross-product of the sets of component modes. We assume that the component transition systems operate synchronously, i.e., for each spacecraft transition every component performs a transition.

A model-based configuration manager uses its transition system model to both identify the current configuration of the spacecraft, called *mode identification* (MI), and to move the spacecraft into a new configuration that achieves the desired configuration goals, called *mode reconfiguration* (MR). MI incrementally generates all spacecraft transitions from the previous configuration such that the models of the resulting configurations are consistent with the current observations (see Figure 9). MR determines the commands to be sent to the spacecraft such that the resulting transitions put the spacecraft into a configuration that achieves the configuration goal in the next state (see Figure 10). The use of a spacecraft model in both MI

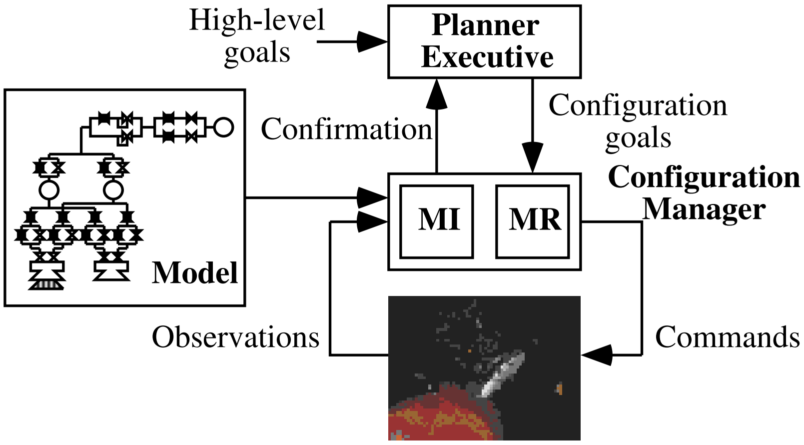

Figure 7: Model-based reactive configuration management.

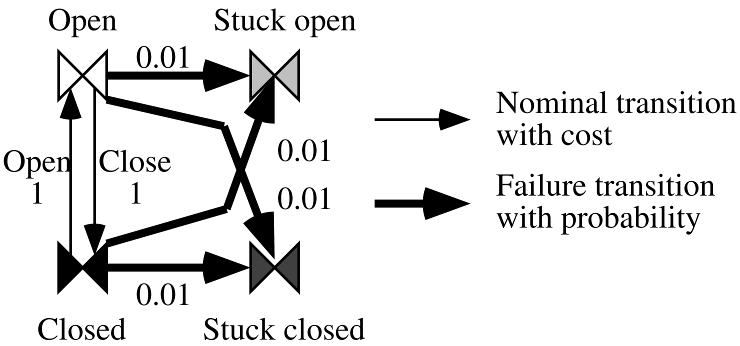

Figure 8: Transition system model of a valve. "Open" and "Closed" are normal operating modes, while "Stuck open" and "Stuck closed" are failure modes. The "Open" command has unit cost and causes a mode transition from "Closed" to "Open." Similarly for the "Close" command. Failure transitions move the valve from the normal operating modes to one of the failure modes with probability 0.01.

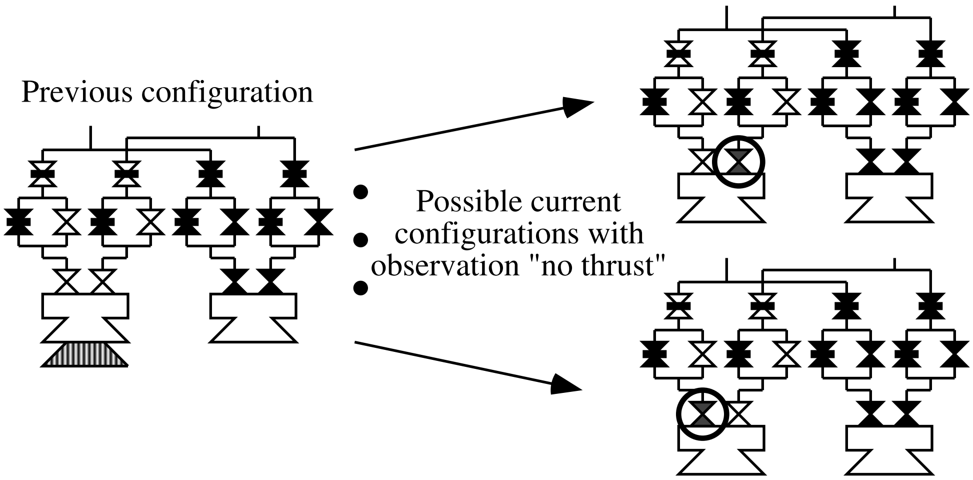

Figure 9: Mode identification. The figure shows a situation in which the left engine is firing normally in the previous state, but no thrust is observed in the current state. MI's task is to identify the configurations into which the spacecraft has transitioned, that account for this observation. The figure shows two possible transitions, corresponding to one of the main engine valves failing "stuck closed" (failed valves are circled). Many other transitions, including more unlikely double faults, can also account for the observations.

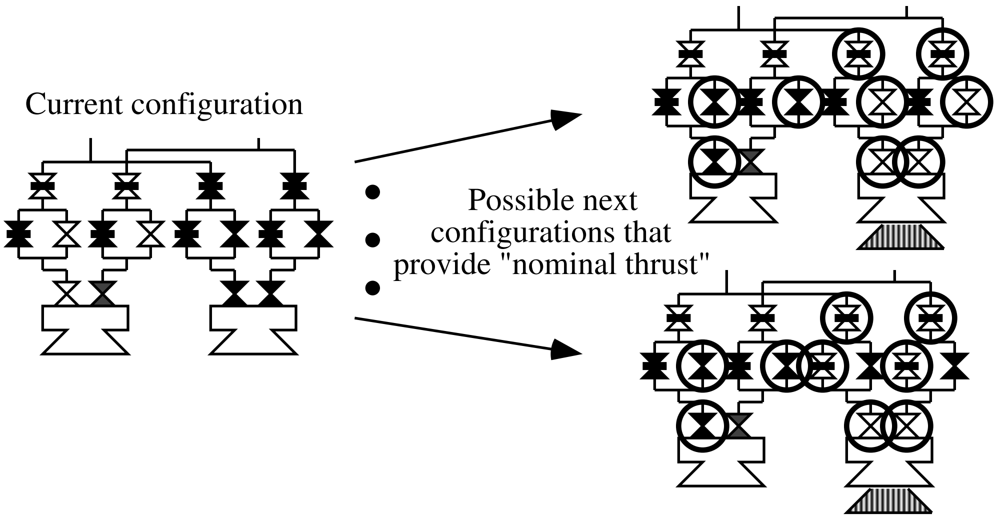

Figure 10: Mode reconfiguration. The figure shows a situation in which MI has identified a failed main engine valve leading to the left main engine. MR reasons that normal thrust can be restored in the next state if an appropriate set of valves leading to the right engine are opened. The figure shows two of the many configurations that achieve the desired goal (circled valves are commanded to change state). Transitioning to the configuration at the top has lower cost since only necessary pyro valves are fired. The valves leading to the left engine are turned off to satisfy a constraint that at most one engine may fire at one time.

Number of components	80
Average modes/component	3.5
Number of propositions	3424
Number of clauses	11101

Table 1: Newmaap spacecraft model properties.

and MR ensures that configuration goals are achieved correctly.

Both MI and MR are reactive. MI infers the current configuration from knowledge of the previous configuration and current observations. MR only considers commands that achieve the configuration goal in the next state. Given these commitments, the decision to model component transitions as synchronous is key. An alternative is to model multiple component transitions through interleaving. However, this can place an arbitrary distance between the current configuration and a goal configuration, defeating the desire to limit inference to a small fixed number of states. Hence we model component transitions as being synchronous. If component transitions in the underlying hardware/software are not synchronous, our modeling assumption is still correct as long as some interleaving of transitions achieves the desired configuration.

In practice, MI and MR need not generate all transitions and control commands, respectively. Rather, just the most likely transitions and an optimal control command are required. We efficiently generate these by recasting MI and MR as *combinatorial optimization problems*. In this reformulation, MI incrementally tracks the likely spacecraft trajectories by always extending the trajectories leading to the current configurations by the most likely transitions. MR then identifies the command with the lowest expected cost that transitions from the likely current configurations to a configuration that achieves the desired goal. We efficiently solve these combinatorial optimization problems use a *conflict-directed best-first search* algorithm. See (Williams & Nayak 1996) for a formal characterization of MI and MR and a description of the search algorithm.

A quick trip to Saturn

Following the announcement of the New Millennium program in early 1995, spacecraft engineers from JPL challenged a group of AI researchers at NASA Ames and JPL to demonstrate, within the short span of five months, a fully autonomous architecture for spacecraft control. To evaluate the architecture the JPL engineers defined the Newmaap spacecraft and scenario based on Cassini. The Newmaap spacecraft is a scaled down version of Cassini that retains the most challenging aspects of spacecraft control. The Newmaap scenario is based on the most complex mission phase of

Scenario	MI			MR	
	Check	Accept	Time	Check	Time
EGA preaim failure	7	2	2.2	4	1.7
BPLVD failed	5	2	2.7	8	2.9
IRU failed	4	2	1.5	4	1.6
EGA burn failure	7	2	2.2	11	3.6
Acc failed	4	2	2.5	5	1.9
ME too hot	6	2	2.4	13	3.8
Acc low	16	3	5.5	20	6.1

Table 2: Results from the seven Newmaap failure recovery scenarios. Time is in seconds on a Sparc 5.

Cassini—successful insertion into Saturn's orbit even in the event of any single point of failure.

The AI researchers, working closely with the spacecraft engineers, developed an autonomous agent architecture that integrates Livingstone with the HSTS planning and scheduling system (Muscettola 1994) and a multi-threaded smart executive (Pell *et al.* 1996b) based on RAPS (Firby 1995). In this architecture (see (Pell *et al.* 1996a) for details) HSTS translates high-level goals into partially-ordered tokens on resource timelines. The executive executes planner tokens by translating them into low-level spacecraft commands while enforcing temporal constraints between tokens. Livingstone tracks spacecraft state and planner tokens, and reconfigures for failed tokens. The above autonomous agent architecture was demonstrated to successfully navigate the simulated Newmaap spacecraft into Saturn orbit during its one hour insertion window, despite about half a dozen failures. Consequently, Livingstone, HSTS, and the smart executive have been selected to fly Deep Space One, forming the core autonomy architecture of NASA's New Millennium program.

Table 1 provides summary information about Livingstone's model of the Newmaap spacecraft, demonstrating its complexity. The Newmaap demonstration included seven failure scenarios. From Livingstone's viewpoint, each scenario required identifying likely failure transitions using MI and deciding on a set of control commands to recover from the failure using MR. Table 2 shows the results of running Livingstone on these scenarios.

The first column names each of the scenarios; a discussion of the details of these scenarios is beyond the scope of this article. The second and fifth columns show the number of solutions checked by MI and MR, respectively. On can see that even though the spacecraft model is large, the use of conflict-directed search dramatically focuses the search. The third column shows the number of leading trajectory extensions identified by MI. The limited sensing available on the Newmaap spacecraft often makes it impossible to identify unique trajectories. The fourth and sixth columns show the time spent by MI and MR on each scenario, once again demonstrating the efficiency of our approach.

The new millennium

We are only now becoming aware of the rapid construction of a ubiquitous, immobile robot infrastructure, that rivals the construction of the world-wide web, and has the potential for profound social, economic, and environmental change. Tapping into this potential will require embodying immobots with sophisticated regulatory and immune systems that accurately and robustly control their complex internal functions. Developing these systems requires fundamental advances in model-based autonomous system architectures that are self-modeling, self-configuring, model-based programmable, and support deliberated reactions. This can only be accomplished through a coupling of the diverse set of high-level, symbolic methods and adaptive autonomic methods offered by AI. While a mere glimmer of Spock and HAL, our two model-based immobots, Livingstone and Moriarty, provide seeds for an exciting new millennium.

Acknowledgements

We would like to thank the many individuals who have influenced this work. Oren Etzioni's provocative 1992 Spring Symposium talk on softbots provided a catalyst. Thanks to the responsive environment team, particularly Bill Millar, Joseph O'Sullivan, and Ying Zhang. Thanks to our Remote Agent collaborators, including James Kurien, Nicola Muscettola, Barney Pell, Greg Swietek, Douglas Bernard, Steve Chien, Erann Gat, and Reid Simmons. Finally, thanks to Yvonne Clearwater for help with the graphics.

References

Agre, P., and Chapman, D. 1987. Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, 268–272.

Brooks, R. A. 1991. Intelligence without reason. In *Proceedings of IJCAI-91*, 569–595.

de Kleer, J., and Williams, B. C. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32(1):97–130.

de Kleer, J., and Williams, B. C., eds. 1991. *Artificial Intelligence*, volume 51. Elsevier.

Dent, L.; Boticario, J.; McDermott, J.; Mitchell, T.; and Zabowski, D. 1992. A personal learning apprentice. In *Proceedings of AAAI-92*, 96–103.

Elrod, S.; Hall, G.; Costana, R.; Dixon, M.; and des Rivieres, J. 1993. Responsive office environments. *Communications of the ACM* 36(7):84–85.

Etzioni, O., and Segal, R. 1992. Softbots as testbeds for machine learning. In *AAAI Spring Symposium on Knowledge Assimilation.*

Etzioni, O., and Weld, D. 1994. A softbot-based interface to the internet. *Communications of the ACM* 37(7):72–79.

Firby, R. J. 1995. The RAP language manual. Animate Agent Project Working Note AAP-6, University of Chicago.

Hamscher, W. C. 1991. Modeling digital circuits for troubleshooting. *Artificial Intelligence* 51:223–271.

Kautz, H.; Selman, B.; Coen, M.; Ketchpel, S.; and Ramming, C. 1994. An experiment in the design of software agents. In *Proceedings of AAAI-94*, 438–443.

Knoblock, C. A., and Levy, A. Y., eds. 1995. *Working Notes of the AAAI Spring Symposium Series on Information Gathering in Distributed and Heterogeneous Environments.*

Levy, A. Y.; Rajaraman, A.; and Ordille, J. J. 1996. Query answering algorithms for information agents. In *Proceedings of AAAI-96*, 40–47.

Maes, P., and Kozierok, R. 1993. Learning interface agents. In *Proceedings of AAAI-93*, 459–465.

Malik, A., and Struss, P. 1996. Diagnosis of dynamic systems does not necessarily require simulation. In *Proceedings of the Tenth International Workshop on Qualitative Reasoning, AAAI Technical Report WS-96-01, 127–136.*

Manna, Z., and Pnueli, A. 1992. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag.

Muscettola, N. 1994. HSTS: Integrating planning and scheduling. In Fox, M., and Zweben, M., eds., *Intelligent Scheduling.* Morgan Kaufmann.

Pell, B.; Bernard, D. E.; Chien, S. A.; Gat, E.; Muscettola, N.; Nayak, P. P.; Wagner, M. D.; and Williams, B. C. 1996a. A remote agent prototype for spacecraft autonomy. In *Proceedings of the SPIE Conference on Optical Science, Engineering, and Instrumentation.*

Pell, B.; Gat, E.; Keesing, R.; and Muscettola, N. 1996b. Plan execution for autonomous spacecraft. In *Proceedings of the 1996 AAAI Fall Symposium on Plan Execution.*

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–96.

Sacks, E. P., and Doyle, J. 1992. Prolegomena to any future qualitative physics. *Computational Intelligence* 8(2):187–209.

Weld, D. S., and de Kleer, J., eds. 1990. *Readings in Qualitative Reasoning About Physical Systems.* San Mateo, California: Morgan Kaufmann Publishers, Inc.

Williams, B. C., and Millar, B. 1996. Automated decomposition of model-based learning problems. In *Proceedings of the Tenth International Workshop on Qualitative Reasoning, AAAI Technical Report WS-96-01*, 265–273.

Williams, B. C., and Nayak, P. P. 1996. A model-based approach to reactive self-configuring systems. In *Proceedings of AAAI-96*, 971–978.

Williams, B. C., and Raiman, O. 1994. Decompositional modeling through caricatural reasoning. In *Proceedings of AAAI-94*, 1199–1204.

Williams, B. C. 1990. Interaction-based invention: Designing novel devices from first principles. In *Proceedings of AAAI-90*, 349–356.

Williams, B. C. 1996. Model-based autonomous systems in the new millennium. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, 275–282.

Zhang, Y.; Williams, B. C.; and Elrod, S. 1993. Model estimation and energy-efficient control for building management systems. Technical report, Xerox PARC.