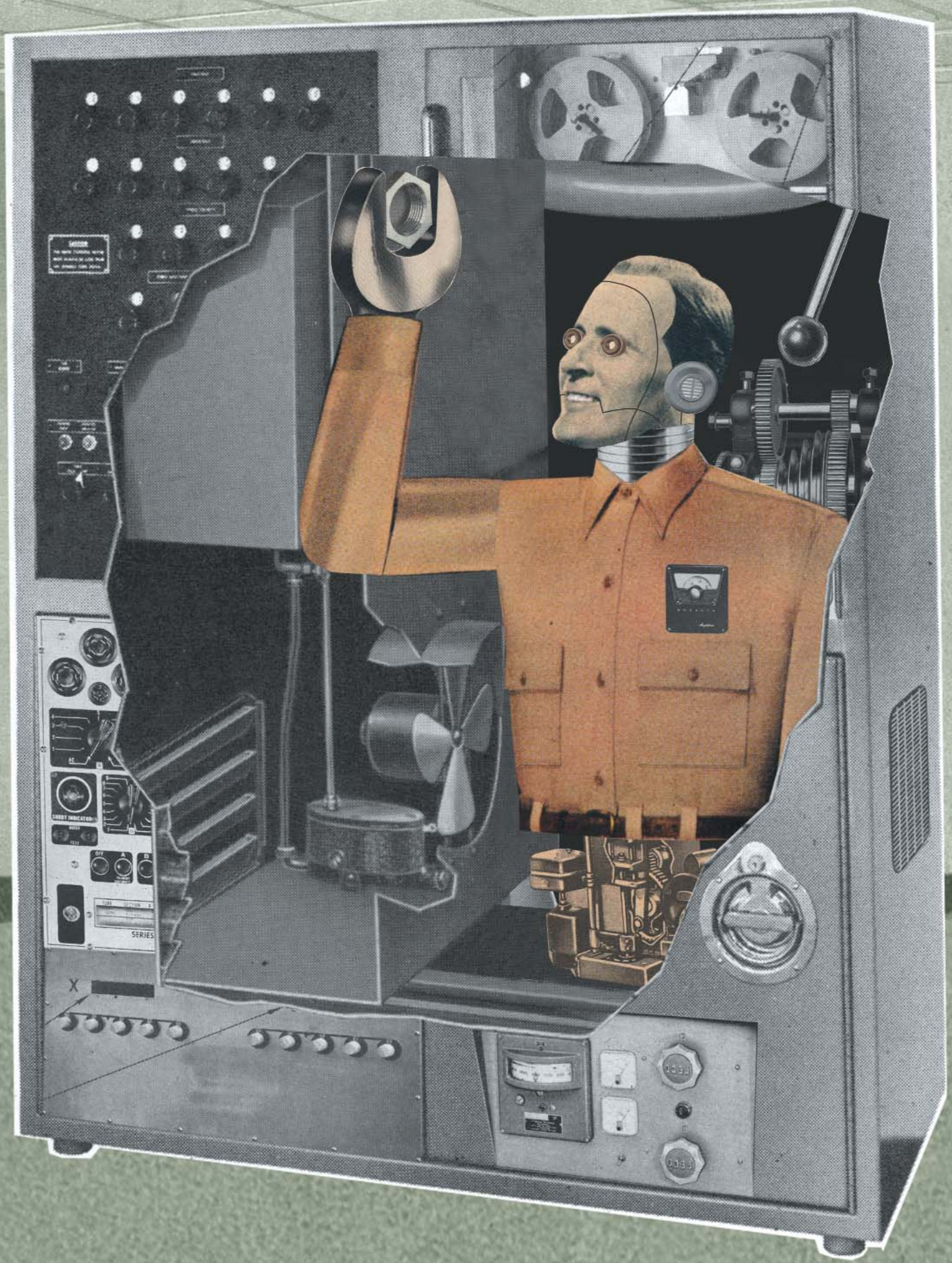

SERIES

immobots take control

IT'S A DANGEROUS WORLD, BUT MACHINES THAT "KNOW THEMSELVES" CAN DEAL WITH CRISES AUTONOMOUSLY—AND OFTEN MORE EFFECTIVELY THAN THEIR HUMAN PROGRAMMERS

BY WADE ROUSH
ILLUSTRATIONS BY DAVID PLUNKERT

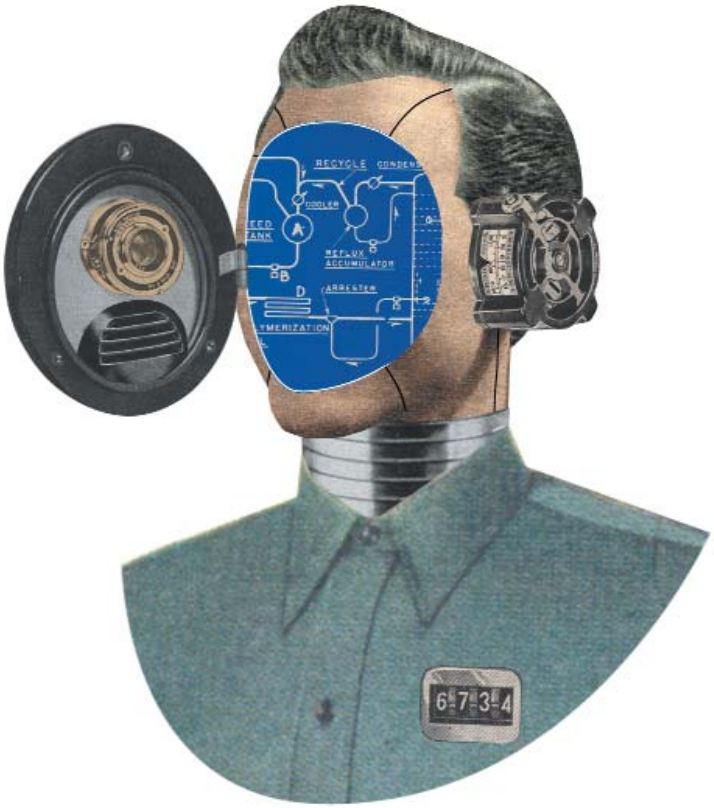
The Mars Polar Lander never had a chance.

At 12:02 P.M. California time on December 3, 1999, after an 11-month journey to Mars, the NASA spacecraft slewed its antenna away from Earth in preparation for entry into the Martian atmosphere. That was the last time mission controllers heard from it. According to the scenario a NASA accident-review board deemed most likely, the Lander dropped out of orbit, deployed its parachute, and began firing its descent engines to slow its fall—just as it was programmed to do. But as the craft's three landing legs automatically unfolded, sensors in the legs sent false signals to the Lander's control software, indicating that it had touched down. Not programmed to deal with such a scenario, the software

ignored signs that the craft was still aloft and, at an altitude of 40 meters, shut down the descent engines. Gravity took over, and the delicate craft slammed into the rocky Martian surface with the energy of a high-speed car crash.

That same year, but millions of kilometers away, another NASA craft dealt with crisis more adroitly. Deep Space One had just begun a turn to take optical readings that would guide a planned flyby of a nearby asteroid. As it turned off its camera to conserve energy, the power switch stuck open. This redirected power needed by other essential components, interrupting the maneuver and threatening to put the spacecraft into a semicomatose "fail-safe" mode that could have taken ground controllers weeks to undo. By that time, the asteroid would have been left far behind.

But Deep Space One had something Mars Polar Lander lacked: an onboard robot able to think autonomously and handle the unexpected. Using its engineering knowledge, the robot tried to repair the switch by toggling it on and off. When this failed, it devised a successful plan to complete the navigation maneuver, and the craft proceeded unharmed.



The robot that saved Deep Space One was in the vanguard of a new breed of machines poised to have a big impact in space and here on Earth. Quite unlike the metallic contraptions that march stiffly through sci-fi movies or the mindless, stripped-down devices that heft parts on our assembly lines, the new robots have more brain than brawn. Each possesses a detailed picture of its own inner workings—encoded in software-based models—that gives it the ability to respond in novel ways to events its programmers might not have anticipated. Because many of these inward-focused, self-reconfiguring machines don't move, some computer scientists call them immobile robots, or "immobots."

Immobots are already beginning to crop up in situations where autonomy is important. They are needed either because direct operator control is impossible (for example, in space probes so distant that radio signals take minutes or hours to reach them) or because humans lack the skill or the desire to oversee all the details (in such down-to-earth systems as office machines, water treatment plants, and internal combustion engines). "There are lots of systems in the world that have sensors and actuators, but don't look like traditional mobile robots," says Brian Williams, a former NASA researcher who coinvented Deep Space One's autonomous software and is now a professor at MIT's Space Systems and Artificial Intelligence Laboratories.

Once programmed with immobotic software, Williams explains, these systems "have a commonsense model of the physics of their internal components and can reason from that model to determine what is wrong and to know how to act." Such systems are more self-reliant than typical computers, which are very good at executing mindless, step-by-step instructions laid out for them by software engineers. However, computers are still amateurs when it comes to thinking their way through unforeseen crises such as component failures.

Immobotic reasoning is turning up in everyday systems, including cars, office copiers, and database software. And if advances in the field continue apace, many of the infrastructure technologies we depend on—heating, ventilation, and air conditioning systems; telephone and computer networks; air traffic systems; and electrical grids—might eventually be run by immobile-robot brains. Indeed, entire cities and regions could be transformed into massively distributed immobots. "The problems where you stare at the computer and scratch your head—that's where sophisticated modeling really has a benefit," says Sam Lightstone, a senior technical-development manager at IBM's Toronto Research Lab. "It allows a computer system to make choices and analyses that a human being couldn't."

model behavior

Deep Space One's grace under pressure marked an early triumph for immobotkind. But in truth, the stuck power switch wasn't an authentic crisis: ground controllers deliberately misled the craft's control software so they could see how Remote Agent, the system developed by Williams and his NASA colleagues, would respond. It was all part of an extensive field test of a programming philosophy known as "model-based reasoning," the core principle of immobot design.

To make machines behave autonomously, most practitioners in robotics and control engineering have long used "heuristic" programs that amount to lists of rules for accomplishing a goal and dealing with contingencies. For example, "If A is true, then do B. If C is true, then do D." The trouble, many artificial-intelligence experts assert, is that traditional, hand-coded software can be either reliable or affordable. Not both.

The Polar Lander mishap and several other software-related failures that marred recent NASA missions demonstrate

that so many of the situations we entrust to software are extraordinarily complex. And human programmers working within tight schedules and limited budgets simply cannot write code that anticipates every contingency. When they try to do so, the software is more often than not so convoluted and slapdash that it contains hidden, potentially fatal bugs *(see "Why Software Is So Bad,"* TR *July/August 2002)*.

A model-based program that reasons like Remote Agent isn't built that way at all. It looks like a picture of the machine it was designed to control, painted in the logical language of computers. Both mobile and immobile robots can use this picture to model themselves and choose the fastest, safest, or most cost-effective way to implement an operator's instructions or deal with an emergency. "The idea is very simple," Williams explains. "Provide the program with a physical plan of the system and let the software deduce what to do."

The key to Remote Agent's real-world reliability, says Robert Rasmussen, chief architect of the Mission Data System project at NASA's Jet Propulsion Laboratory, was its collection of very simple models. Each model defined one of Deep Space One's mechanical and electrical components in terms of its possible states. A valve, for example, might be represented by one of two operating modes, "open" or "closed," and one of two failure modes, "stuck open" or "stuck closed." Mathematical rules outlined the possible transitions between modes and the probabilities associated with each one. It's very unlikely that a valve would go from stuck open to stuck closed, for example, so the software knew that it should spend less time investigating such possibilities in the case of a failure.

At the software's higher levels, hundreds of component models were strung together according to the spacecraft's blueprints. In actual operation, the software would begin with no more than a general goal provided by operators, as well as a picture of the spacecraft's current state as indicated by sensors that monitor each valve, relay, gyroscope, fuel tank, and camera. Building from its knowledge of the craft's innards, the software would create a step-by-step plan for reaching the goal or working around problems.

The advantage of this kind of programming is that software developers don't have to lay out every detail of an operation—or imagine and prepare for all possible mishaps. "Engineers thinking about every bad thing they can think of and making sure the spacecraft can respond to those situations is a very time-consuming chess game," says Rasmussen. Model-based programming, by contrast, "provides us with a way to make systems behave the way we want, without spending years and millions of dollars."

office autonomy

A deep-space probe obviously requires much more autonomy than, say, a photocopier. But heavily used office machines must meet a similar demand for reliability and efficiency—especially when the boss asks for 30 bound copies of the annual report for a board meeting that starts in three minutes. That's why engineers at Xerox and its recently spun-off Palo Alto Research Center (PARC) have begun to build immobot intelligence into high-end copy machines.

Like a big-city airport with flights arriving and departing on shared runways, a copier's biggest challenge is scheduling: it needs to launch the next sheet of blank paper into the printing system as soon as the previous one is out of the way. But as Xerox's previous generation of copiers and printers became increasingly complex, says Daniel Bobrow, an artificial-intelligence researcher in PARC's Systems and Practices Lab, the company noticed that the process was taking more and more time, especially when users selected options such as two-sided copying, sorting, and stapling.

What's more, the machines' heuristic scheduling software lacked full understanding of the interactions among stations, so it treated each of the processes as separate tasks, all of which had to be completed before the next sheet could enter the work path. "The only way to anticipate every possible request was to take any request that was different from the ones [programmers] had thought of and break it down into two or more jobs," Bobrow explains. Even worse, the scheduling software had to be substantially rewritten each time Xerox wanted to add new features to its machines.

THE IDEA BEHIND IMMOBOTIC CONTROL IS VERY SIMPLE: PROVIDE THE PROGRAM WITH A PHYSICAL PLAN OF THE SYSTEM AND LET THE SOFTWARE DEDUCE WHAT TO DO.

But the DocuColor 2045 and the DocuColor 2060, Xerox's flagship printer-copiers, have taken a big step toward sentience. Both truck-size $90,000-to-$120,000 machines come with model-based scheduling programs, which PARC researchers designed to optimize moment-to-moment paper flow. "You want to have a number of sheets going through in succession, and when you have two-sided copies coming back around, you want them to be interspersed with things doing the first side," says Bobrow. "So you have to actually look at the timing of these various things and build a model of what's going on."

Say you're making 70 copies of a booklet. Even before you press the Start button, a machine running PARC's model can predict that stapling will be the slowest part of the job and communicate this fact to other components of the machine, allowing them to run concurrently and without creating a big backup. Another advantage, Bobrow says, is that the DocuColor controller models are built from smaller models hard-wired into each component. When a new station such as a scanner or sorter is added, it transmits its internal model to the central controller, providing a painless software upgrade.

"Model-based programming is doing a tremendous job for us in terms of improving [copier] productivity," says Bobrow. Owners of the machines can add and remove components as they like, and they can let the machines toil unattended for hours. What's more, he says, Xerox can use its current models

to simulate new equipment configurations and "evaluate how things would work" before investing a dime in physical prototypes of its next-generation machines.

"This distinction between telling a system how to do its job and telling the system the end result you want is very fundamental," says Robert Morris, director of IBM's Almaden Research Center in San Jose, CA. IBM is working to build what it calls "autonomic" characteristics—model-based features, as well as others that employ classic heuristic programming—into products such as Web servers and storage networks. These features will allow the products to reconfigure themselves for optimal performance, depending on what's being asked of them.

One project, for example, involves building IBM's DB/2 database software with models that allow databases to learn from past queries and suggest ways to rewrite new ones to retrieve data faster and more precisely. Simple efficiency is dictating this move, Morris says. "We spend so much of our time managing these systems and so much of our money paying people to run them. We'd better stop spending so much on the tedium and more on the new technologies."

hitting the road

If handling unforeseen situations is important for a copy machine, it will be absolutely crucial for machines like the processor-heavy automobiles of the near future *(see "The Networked Car,"* TR *September 2002)*. According to MIT's Williams, officials from Toyota USA recently visited the Artificial Intelligence Laboratory looking for ways to reduce to zero the failure rate of the more than 30 computer processors inside each Toyota vehicle. "At that level," says Williams, "control systems need to have the reasoning and smarts of a robot." And although engineers are still decades away from putting an immobot into the driver's seat—a concept that car buyers may never accept anyway—monitoring a car's internal functions is a very different matter.

Drivers can't spare the attention and don't have the skills to sense problems with fuel pressure, emissions levels, antilock brakes, and any of the other complex systems that influence a car's performance. Modern passenger vehicles contain dozens of processors, so-called electronic control units, that monitor and control these functions with conventional software. But researchers in Europe are already developing onboard diagnostic immobots that run on the same control units and respond to internal automotive failures as soon as (or even before) they occur.

Traditional heuristic programs fall short in such situations because they "fail to capture the first-principles knowledge human experts use" to diagnose problems and plan repairs, says Peter Struss, a computer scientist at the University of Technology at Munich. By "first-principles knowledge," Struss means understanding of the structure, behavior, and interactions of valves, pipes, sensors, and other auto parts. At Occ'm, the Munich software company he cofounded, Struss has spent the last several years building and testing software models of auto parts in collaboration with companies such as BMW, Fiat, DaimlerChrysler, Peugeot, Citroën, and Renault. He's finding that his models identify problems that sometimes elude even expert mechanics.

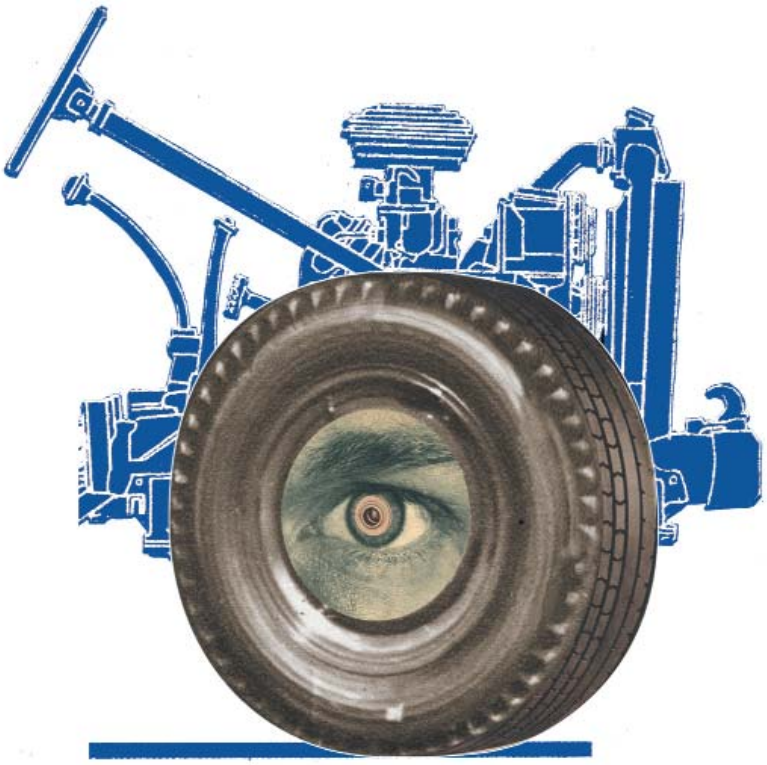
Say the air conditioning in your vehicle won't work. An onboard immobot might quickly deduce that the problem is a malfunctioning fuel-level sensor in the gas tank. "What's the interdependency?" Struss asks. In some cars, he explains, "the AC control system has to ask the engine control unit whether, as a consumer, it's allowed to come on. The engine management system will check whether there's enough fuel. And if not, it will deny the request."

A model-based diagnostic system knows such details in advance. Therefore, if you've got a full tank and the air conditioning still won't work, the model-based diagnostic makes an educated guess at the cause. "Workshop people can read out the diagnostics from control units and see six or seven trouble codes, but it may not be obvious what the ultimate cause is," says Struss. "They cannot see all the interactions." But an immobot can.

In two prototype self-diagnosing vehicles Struss helped build for a project of the European Commission, the immobot software ran on laptops inside the passenger cabins. However, Struss predicts that as early as 2004, similar model-based programs will be built into new cars' onboard diagnostic modules. Meanwhile, he is developing programming tools to help automotive designers build cars with diagnostic systems in mind, rather than (as is the case today) adding them as an afterthought. This means, for example, making sure sensors are always placed where they'll return the data an immobot needs to identify—and someday soon, correct—component failures.

TO MECHANICS WORKING ON A CAR, IT MIGHT NOT BE OBVIOUS WHAT THE ULTIMATE PROBLEM IS. THEY CAN'T SEE THE INTERACTIONS BETWEEN SYSTEMS, BUT AN IMMOBOT CAN.

The focus of Struss's work and related studies in Europe—including a project at France's Centre National de la Recherche Scientifique to build a model-based system that will help garage mechanics isolate faults in electronic circuits—indicates that when immobot software finally makes its mass-market debut, it will almost certainly be in the automotive realm, says computer scientist Louise Travé-Massuyès, director of the French project. "The major problem in automobiles," she says, "is the increasing complexity of electrical systems, which means that companies are looking for intelligent tools to help them analyze their products." The key economic and safety advantage of model-based diagnosis, she argues, is that "faults and symptoms do not need to be anticipated." In other words, software designers need only spend their time learning how a car's systems should act, not how they might go wrong.



colossal control

Three technologies in which immobots are already taking hold—spacecraft, copiers, and cars—are roughly on the same size scale as the human body. But as Struss is demonstrating in another project, there's nothing preventing immobots from tackling challenges much larger in size and scope.

In Porto Alegre, Brazil, an industrial metropolis with a population of about 1.3 million, the water utility lacks qualified operators to run all five of the city's water treatment plants. Struss says that many of "the people looking after the drinking water have limited education about how to do that." To support the operators, Struss and his university colleagues are collaborating with the utility to build an advisory immobot that will monitor such water quality measures as turbidity, color, acidity, and alkalinity. The goal is to spot trouble as early as possible and automatically propose therapies.

Although making water safe for human consumption may seem a simple matter, Struss says that it is much more difficult to translate that task into models than it is to deal with the workings of an automobile. That's because a reservoir is more an ecosystem than a machine, and that means that modeling "cannot be limited to looking for faulty components, but [must also look for] unanticipated influences and interactions," he explains.

If there's a change in sensor readings, for example, an immobot-controlled water-treatment plant can reason its way back to the most likely cause only if it has enough interlinked models of the physical and biological processes that affect water quality. Say the water's concentration of iron is hazardously high. The system should hypothesize that an algal bloom is to blame because dying algae change the acidity of water, which in turn redissolves iron in the sediments. Ideally, under such conditions, the immobot would advise operators about short-term and long-term responses: treating the water immediately to remove the iron and investigating the most likely cause of the algal bloom, excess nutrients from fertilizer in agricultural runoff.

A prototype advisory system is being tested in Porto Alegre. But building complete and accurate models of processes rather than just parts—and making all those models work together—remains a bit beyond the state of the art, Struss says. It's "difficult because you cannot enumerate all the components. Sometimes you don't know what they are."

While these challenges are being worked out, immobot software is making its way into smaller, self-contained systems such as copiers and cars. Before large-scale infrastructure technologies can be endowed with model-based reasoning abilities, however, researchers must overcome another barrier: the skepticism of old-school engineers who are accustomed to keeping their machines on the short leash provided by heuristic control software.

At NASA, for example, cautious mistrust of truly autonomous software almost killed Remote Agent even before Deep Space One left the ground. "Mission managers are by their nature extremely risk-averse people," says Ken Ford, a computer scientist who directs the Institute for Human and Machine Cognition at the University of West Florida. Ford, who was associate director at NASA Ames Research Center at the time Williams was developing Remote Agent, says, "In fact, it was hard to get them to fly it at all, even as an experiment."

But the success of Remote Agent, along with other demonstrations of model-based reasoning, may be slowly swaying the opinions of even conservative engineers. The Jet Propulsion Laboratory's Rasmussen, for instance, is working with Williams's group to develop a model-based program that has been tentatively designated as the main control software for the Mars Science Laboratory, scheduled for launch in 2009. In Edinburgh, Scotland, a small company called Intelligent Applications is working with General Electric International to sell its model-based software for monitoring and diagnosing the behavior of power-producing gas turbines. And the European Commission has launched a project to develop model-based failure-management software for commercial aircraft.

Eventually, researchers say, immobots could become pervasive, helping to control some of our most important infrastructure technologies. In air traffic management, for example, Williams suggests building immobotic systems that would use sensor data from satellites and ground stations to assess local weather conditions, automatically identify and select the safest, most efficient flight paths, and redirect air traffic.

"At the grand scale," Williams says, "you can address a set of problems people have never tackled before." If he and his colleagues are right, the only way to make infrastructure technologies autonomous without increasing the risk of massive software-related failures may be to program them with distinctly human qualities, such as the ability to plan ahead and solve problems creatively. And in a world that seems increasingly dangerous, knowing that immobots are looking out for themselves—and for us—could be a source of comfort. TR