WHATEVER THE REASON FOR THE SOFTWARE FAILURE, WE WOULD LIKE THE SOFTWARE TO BE ABLE TO RECOGNIZE THAT IT HAS FAILED AND TO RECOVER FROM THE FAILURE.

Adding dynamic intelligent fault awareness and recovery to running systems enables the identification of unanticipated failures and the construction of novel workarounds to these failures. Our approach is pervasive and incremental. It is pervasive in that it applies to all components of a large, complex system—not just the "firewall" services. It is incremental in that it coexists with existing faulty, unsafe systems, and it is possible to incrementally increase the safety and reliability of large systems. The approach aims to minimize the cost, in terms of hand-coded specifications with respect to how to isolate and recover from failures.

There are many reasons why software fails, the most common include:

- Assumptions made by the software turn out not to be true at some point. For example, if a piece of software must open a file with a given path name, it will usually succeed; but if the particular disk that corresponds to the path name fails, the file will not be accessible. If the program assumes that the file is accessible, the program will fail. In highly constrained situations, it is possible to enumerate all such failures and hand code specific exception handlers—and such is the standard practice in the industry. In many cases, however, particularly in embedded applications, the number of ways the environment can change becomes so large that the programmer cannot realistically anticipate every possible failure.
- Software is attacked by a hostile agent. This form of failure is similar to the first one except that change in the environment is done explicitly, with the intent to cause the software to fail.
- Software changes introduce incompatibilities. Most software evolves during its lifetime. When incompatibilities are inadvertently introduced, software that previously did not fail for a given situation may now fail.

Whatever the reason for the software failure, we would like the software to be able to recognize that it has failed and to recover from the failure. There are three steps to doing this: noticing the software has failed; diagnosing exactly what software component has failed; and finding an alternative way of achieving the intended behavior.

Approach

In order for the runtime system to reason about its own behavior and intended behavior in this way, certain extra information and algorithms must be present at runtime. In our system, these extra pieces include models of the causal relationships between the software components, models of intended behavior, and models of correct (nominal) execution of the software. Additionally, models of known failure modes can be very helpful but are not required. Finally, the system must be able to sense, at least partially, its state; to reason about the difference between the expected state and the observed state; and to modify the running software (for example, by choosing alternative runtime methods).

Building software systems in this way comes with a certain cost. Models of the software components and their causal relationships, which might otherwise have existed only in the programmer's head, must be made explicit; the reasoning engine must also be linked in to the running program, and the computational cost