Self Adaptive Software

Howard Shrobe & Robert Laddaga

Artificial Intelligence Laboratory
Massachusetts Institue Of Technology
Cambridge, Massachusetts 02139

http://www.ai.mit.edu

The Problem: Can we find a way to build software systems which can autonomouly adapt to unforeseen and changing circumstances? What are the organizing principles for such systems? Is there a common infrastructure that can support all self-adaptive software systems.

Motivation: We are concerned with the problem of building mission-critical systems that operate non-stop in a rapidly changing, and possibly hostile environment. It is unacceptable for these systems to "shatter" when confronted with change; they must instead fluidly adapt to circumstance by: figuring out what is wrong, identifying an alternative approach to getting the job done, establishing a consistent state from which to begin the recovery and then get the job done as well as possible. Reporting a bug and waiting for a new system build is simply not a viable option.

Previous Work: Our project involves many tasks for which there is prior work. In particular, work by Davis and Shrobe and Williams and deKleer on model-based trobleshooting is relevant to the diagnostic component of the task, work on the design of dynamic object oriented languages in the Lisp tradition also provide valuable insights. Work in the programmer's apprentice project by Shrobe, Rich and Waters provides many of the underlying representations needed for this task.

Approach: Our approach to building adaptive software is based on the notion of a Dynamic Domain Architecture (DDA). Dynamic Domain Architectures structure an application domain into layers of common services where each service has a number of variant implementations tailored to different environmental conditions. The architectural level of description also provides "purpose links" which explain how the components of service achieve its overall goals. The DDA development environment synthesizes run-time sentinels to monitor the preconditions of the purpose links. The runtime services of the DDA are invoked when a sentinel signals the failure to achieve an expected condition; the runtime services are responsible for diagnosis of the failure and for selection and execution of a repair procedure. Since the DDA provides many alternative implementations of each component, a typical repair involves rolling back to a recovery point and invoking an alternative implementation.

One of the critical features of the runtime infrastructure of a Dynamic Domain Architecture is its ablility to self diagnose. We are working on reformulating earlier work on model based troubleshooting so that it can be applied to failures in software systems. Model based troubleshooting reasons backwards from symptoms, the failure of a system to behave as expected, to an underlying cause, a description of how a specific component of the system failed to behave correctly. Such failures can be due to failures of the underlying operating system, network or computing hardware, or to an incompleteness in the code's coverage of the phenomenon (as is often the case in image understanding, for example). Reasoning about the behavior of a software system is notoriously difficult, but it can be made easier if the system gathers traces of its behaviors at many places. Runtime diagnosis is, therefore, partly enabled by code transformations in the development environment which synthesize data collectors, error detectors, condition handlers and restart locations. We are working on developing this infrastructure. Once a diagnosis is made, the system must reconfigure itself into a consistent state appropriate for the restart. This requires significant cleanup activity which would be much simpler if the memory of the main behaved transactionally; we are investigating techniques for making transactional memory and studying whether these techniques are appropriate for our needs.

Difficulty: This project is attempting to develop an infrastructure from an entirely new family of software systems. Although each of the components of this infrstructure has its precedents, the overall ensemble is entirely novel and represents a major challenge.

Impact: Increasingly computer systems are designed to be embedded in and interact with their environments and to function over long periods of time. Such systems will experience variations of conditions which are impossible for the designers to explicitly anticipate, nevertheless the systems must be prepared to deal with cope with these circumstances. We currently have no general methodology or infrastructure adequate for this task. If our project succeeds, it will revolutionize the way such systems are designed and implemented.

Future Work: We plan to develop the core infrastructure over the coming year and then begin to develop serious test applications to stress test and shape our framework. We are particularly interested in applying this to image understanding tasks.

Research Support: This work is funded by DARPA under contract number F30602-97-2-0013, administered by Rome Laboratory.