Self-updating Software: An NTT Research Project 9807-15

Proposal for 1999-2000 Funding

Barbara Liskov and Daniel Jackson

Loading and updating software has become a dominant cost in the administration of PCs -- some estimate $5k/machine-year -- and a major source of fragility and unreliability. It is also the key obstacle that stands in the way of pervasive computing: the environment of the future in which large numbers of small devices will operate and update themselves without human intervention.

Our research aims to find a systematic and practical solution to this problem. Our current approach has two prongs.

At one end of the spectrum, we are investigating self-updating in the realm of PC-like devices, in which updates come both from the desires of users for new functionality (in "pull mode") and from manufacturers who offer versions that add new features and fix old problems (in "push mode"). The configurations on the devices are large, complex, and heterogeneous: the same application may well be installed in different configurations on different machines. The updating infrastructure cannot assume total control over the system's configuration, but must compensate automatically for changes made by the user, such as accidental deletion of files.

At the other end of the spectrum, we are investigating self-updating in the realm of embedded devices, such as elevator controllers, in which updates come primarily from the manufacturer. The configurations on the devices are smaller and less complex, and may be assumed to be homogeneous; in this respect the problem is easier. On the other hand, "hot updates" must be possible, in which normal operation is not noticeably interrupted by the updating, and in which state is retained across updates, appropriately transformed for the new code. Demands of reliability and performance are higher, and the scheduling of updates across large numbers of networked devices must be addressed.

Until now, our work has focused on developing the models and architecture that underlie these infrastructures. We have developed a dependency model in which the essential structure of a software system is represented as a graph over components, and a scheme for hot updates

that involves filters that identify objects to be updated and transformers to execute the updates. We plan to complete this work in the course of the next year, and to move into a new phase in which we construct implementations and begin to evaluate their utility.

In the realm of PC-like devices, we plan to focus on the consistency problem: ensuring that configurations are consistent, and generating appropriate actions to establish, maintain and restore consistency. The initial infrastructure will consist of a centralized database that stores information about components, and on the local machine, a registry that stores configuration data and a utility that performs installations and repairs configurations automatically. We will begin with a rudimentary server database, and focus on the automatic management of configurations on the local machine. By representing dependence information as simple ASCII files accompanying components in a compressed archive, we hope to be able to wrap existing components and demonstrate the infrastructure on standard packages, from small shareware programs such as Ghostview and Winzip, to large application suites such as Office. We expect to refine our dependence model as we learn more about the structure of downloaded packages.

In the realm of embedded devices, we plan to focus on the problem of hot updating. We plan to implement on top of Thor, our object-oriented database system that provides persistence and transactions. Using Thor will allow us to leverage our previous work, allowing us to provide an object store in which persistent objects can be upgraded automatically. We plan to look at a number of difficult but important problems that are likely to arise in hot updating, such as how to interleave computation and updating without undesirable interference, how to postpone updating until absolutely necessary, and how to accommodate changes that are not backward compatible.

By the end of 1999, we plan to have completed:

* the foundations: a dependence model and a scheme for hot updates;
* prototype implementations of the two infrastructures;
* a demonstration of self-updating in the PC realm for small packages;
* a demonstration of self-updating in the context of a persistence object store.

Although the two prongs of our project focus initially on different aspects of the problem -- consistency and hot updates respectively -- we believe that, in the long-term, an updating infrastructure for any device must embody solutions to both. Our plan during the continuing support is to develop an implementation that solves the entire problem, together with a set of tools that help system administrators to define consistent updates that move from one well-defined configuration to another.