Self-Updating Software
9807-12&26

Proposal for 1998-1999 Funding

Barbara Liskov and Daniel Jackson

Summary

Self-updating software updates itself at runtime, either autonomously or under the control of a remote authority. We plan to develop the basic infrastructure that is needed to make self-updating software efficient, economical and safe. Our approach is distinguished by the fine granularity of the updates, the flexibility of the mechanism, and the separation of online and offline analysis.

We believe we can make exciting, concrete progress in the short term by building on our previous achievements. In the long term, our project will be a source of fascinating research challenges, and will shape the building blocks of a new generation of software.

Why this problem?

Most research in software has focused on its execution, but increasingly the biggest costs are in its installation, upgrading, and removal. Current methods for installing software are inefficient and unsafe. They consume vast resources, not only in network bandwidth and unnecessary local storage, but, worse, in the attentions of the user.

As software production grows, and as small computing devices with limited memory become more pervasive, the pattern of updates will change from occasional, large updates to frequent, small ones. Moreover, our environment will become filled with software-enabled sensors and actuators whose interfaces allow no direct human control, and thus cannot be updated by current methods at all.

What's the long-term vision?

Our infrastructure will make it easy to build software that updates itself. In response to changes in the environment, to new requests from users, or to publication of new versions, the software will autonomously detect the need for an update; locate the right server; download the new software; certify that the update will have no bad effects; and integrate the new code. All this will occur without major interruptions in service, and will scale to a worldwide network.

Updates will be extremely fine-grained, often at the level of an individual object. The infrastructure will provide simple, reliable and flexible mechanisms, on top of which application developers can implement different updating policies. In some contexts, a 'push' policy will be needed, in which a server makes the decision to perform the update, and acts on objects distributed in perhaps thousands of computing devices. In other contexts, a 'pull' policy will be better, where clients initiate the updating.

Just making this happen is itself a major research challenge. Equally important, however, will be safety: the update will usually be required to preserve existing functionality and state, and must never compromise security or reliability. For embedded devices in particular, the viability of the entire scheme depends on its safety.

Motivating Examples

* Adaptive Front End for Centralized Service. The user buys a service such as a filtered newsfeed, home banking, or airline reservation; as the service is expanded by its provider, the front-end application adapts automatically and invisibly, so the user sees new features as they become available.

* New Features for Embedded Devices. An elevator manufacturer improves its scheduling algorithm; new software is automatically installed at local plants worldwide. A new telephone switch feature requires additional computation at the handset; when the user requests the feature, the relevant software is downloaded. To optimize traffic flow in a city, the transport authority decides to price roads dynamically, so that the charge for travelling on a particular stretch depends on how congested it is at that time; car navigation software then adapts invisibly to a new price information source. A cable TV company offers a new way to choose programs according to user profiles; the user's VCR downloads new software spontaneously.

* Embedded Devices Adapt to Changing Environment. An elevator detects heavy load patterns and downloads software to plan better. A climate-control system adapts to a heatwave by downloading a special energy-saving regimen. A PABX detects the addition to a local network of a new device for teleconferencing, and fetches appropriate software.

* Gentle-Slope Application. The initial installation of a program by a user gives a minimal system with extensive help facilities; as the user matures, the application automatically fetches new features from a server, removes basic help functions, and morphs the user interface.

What are the major research issues?

Two key problems need to be solved. First is the object management problem (1). How is the need for an update perceived, and how are the updates performed? How is the storage of objects organized? Where do objects reside in the network, and how are they named? How and when are objects garbage-collected?

Second is the certification problem (2). How are objects checked in advance of their integration into the running client system, in order to ensure that the update is safe? What kinds of properties can be checked, and how are they specified? Are specifications predetermined, or generated according to circumstance? Is an object checked against one specification or several? Are specifications given per class or per method?

Although these two problems are largely independent, we expect some coupling: how and where specifications are stored, for example, will affect what kinds of certification schemes make sense.

This is a high risk project. It is hard to predict how the computing environment will look 10 years from now, so we are wary of preconceived solutions. Nevertheless, we have some initial ideas about how to approach these problems. These are simple and speculative first steps; obviously, we hope to develop more powerful and general answers to these questions.

(1) The update protocol can be phased to minimize its runtime cost. When a server decides to update a class of objects, it will use directory and caching schemes (as in Thor) to find out where affected objects reside. Individual sites then identify the affected objects locally, and modify them temporarily (by inserting new dispatch vectors) so that subsequent uses are trapped. When a call to a method of an affected object occurs, a filter is applied to the object. If the object does not pass the filter, it is not updated (and its old dispatch vector is reinstated). Otherwise, new code for the object is downloaded from the server database, and a new object representation is made that incorporates the old data, but points to the new code. With filters, the decision to update can be fine-grained: based not only on gross properties of the environment in which the object operates, but also on the local state of the object itself.

The above describes the mechanism to support a push policy. For a pull policy, the site where the object resides might intercept a call to an unsupported method - perhaps using Java's reflection mechanism - and fetch the new object autonomously from the server database.

(2) Our certification approach combines offline and online activities. Performing checking offline reduces the runtime cost, but is less safe. An economical balance is called for, in which compute-intensive analysis is performed offline for establishing more subtle properties, and simpler, but still critical checks are performed online. We plan also to accommodate analyses that cannot be automated at all; these might be carried out manually offline, and then certified cryptographically.

The simplest form of online checking will be little more than type checking, either implemented conventionally within a just-in-time compiler, or more tightly interwoven with the running program using reflection. The simplest form of offline checking will be an elaborated type analysis to establish that certain exceptions (such as failed downcasts) will never be thrown. But much more ambitious schemes are possible. Mutability information - which methods write which fields, for example - might be used to approximate subtype compatibility. Using object modelling notions, we might characterize methods in terms of how they add, delete and change links to other objects, and check that such behaviour is preserved. It might also prove useful to track dependences; updates of one object may be correlated with updates of another.

As our research progresses, we will address the many obstacles in the way of making self-updating software practical. Many systems, for example, will need some mechanism that gives users some control over how updates are made. What kind of abstractions and interfaces are suitable remains to be seen.

What's the short-term plan?

We have two initial goals. The first is to get a prototype running, to give us a solid basis on which to experiment, and to give us early feedback on some simple schemes. The second is to develop a suite of touchstone examples to guide the work. Particularly in this latter goal, we hope to work with NTT researchers: to learn about what kinds of problems they see as most critical in practice, and perhaps to take NTT examples as challenge applications to be supported by our infrastructure.

We understand that Dr. Naito (of NTT Software Labs) has started a group with similar goals, and we are eager to exchange ideas. We also look forward to talking to Dr. Horita (of NTT Software Labs), and to the members of the NTT Optical Network Systems Labs who have expressed interest in our work.

In order to make rapid progress, we plan to build on our existing achievements. In particular, Thor will provide much of the mechanism for storage and naming of objects. Using Thor as a platform will allow us to focus immediately on some of the more unusual and radical aspects of the problem.

Why us?

Liskov has extensive experience in all the fundamental areas of this project: in asynchronous distributed systems, in object-oriented compilation and runtime support, and in typing and subtyping disciplines. Her focus will be primarily on the object management aspect.

Jackson has been working on practical and tractable specification techniques, and technologies for fully automatic analysis of code and specifications. He will therefore focus primarily on the certification aspect.