Mobile Computation

(Position Paper)

Luca Cardelli

Digital, Systems Research Center

1 Introduction

These are thoughts about some new issues on the computing horizon. I believe a fundamentally new model of computation has been brewing in the shadow of the World Wide Web, and very little foundational work exists to back it up. The foundational work that exists does not directly apply, although hopefully much current knowledge can be adapted.

Looking back a few years, we may notice that we finally got rid of assembly languages. How did that happen? It was partially because of improvements in complier technology and hardware speed, but mostly because assembly languages are not *compile-time portable*. That is, one simply cannot recompile assembly code for a different architecture. Moreover, new architectures have been emerging faster than one can recode.

As an interesting parallel, then, notice that none of the currently popular languages are *run-time portable*. Even though programs can be recompiled for different architectures, one cannot take a running program and port it to a different architecture while the program is running. This, however, is precisely what must happen with computations over the Web, where connections to unknown architectures are established faster than one can recompile. Existing languages and compiler technology are not well suited to this kind of environment.

There is a certain inevitability that comes with exponential growth. Out of the dramatic expansion of the Web will inevitably emerge a new generation of languages with radically different implementation characteristics but also, of more interest to us, with unusual semantic properties. What is the meaning of taking a running computation and moving it to an unknown network site? In most current languages, it makes very little sense. But, in order to program the Web, we must invent languages and semantics where this kind of activity makes sense.

2 Programming the Web

The Web is the single global computer we use to read about in science fiction novels. Now we have it; how do we program it? Apart from the general intellectual challenge, we need to understand the intended application domain, since there are already commercial interests involved.

The commercial players are the *content providers* (any business who wants to showcase and sell a product) and the *clients* (anybody with an Internet connection and a

credit card). Content providers want to present their products to clients according to their own unique, highly customized look. To improve the speed and quality of client interaction, they need to download code which runs on client machines. Moreover, they want this code to keep interacting with their servers (i.e., this is not the one-time download of a demo program). Clients, on the other hand, want to interact with multiple servers simultaneously, and combine and compare the answers they are getting. All this means not only a lot of communication, but also frequent and sophisticated movement of computation over the network: virtually at every mouse click.

Until now, content providers have not been able to adopt this model. Their options include setting up areas on America Online, Compuserve, and Bulletin Board Systems. They use the standardized slow-round-trip interfaces provided by those services, or write their own customized gateways at great expense.

Already, the World Wide Web hypertext facilities are changing all that, with content providers opening shop on the internet in large numbers. But hypertext is not programmable, and does not support well the *interactive content* that both content providers and clients require.

Because of the limitation of hypertext, mobile computation promises to be a strong attraction for content providers, and a crucial component for making the Web *the* place where electronic commerce is conducted. Otherwise, the Web may remain simply a browsing tool, with commerce being conducted out of centralized services (which, anyway, will eventually have to come up with their own mobile computation solutions to improve client interaction.)

3 How Computation Moves

There are at the moment four relatively distinct implementation models for moving code and computation. They differ in what kind of entities are transmitted over the network.

- Moving text (represented by Tcl [3]). Source text is sent over the network and interpreted remotely; a good property is that text is architecture independent. This is the simplest and least "connected" model: when text moves, connections that the computation had at the originating site vanish, and must be reestablished at the destination site. Most languages based on this model have poor semantic properties: typically the meaning of a piece of text depends unpredictably and unsafely on the environment in which it runs.
- Moving bytecode (represented by Java [2]). Bytecode is processed (compiled) code but which is still architecture-independent. The processing can establish certain important guarantees, both at the server and client end, and improve performance of transmission and execution. Still, bytecode is just another way of presenting program text: it is inherently disconnected. (Java supports comfortably one continuing connection between a client and a server.)
- Moving closures (represented by Obliq [1]). Not just code, but also the context

in which the code operates is transmitted. The pair of code and dynamic context is called a closure. The context may include already established network connections, which are preserved on transmission. Therefore, live, active, computation can move, and their intrinsic meaning is preserved upon transmission.

- Moving agents (represented by Telescript [4]). Agents are similar to closures in that they carry their context with them as they move from location to location. Agents, however, are meant to be completely self-contained and resource-limited. They do not communicate remotely with other locations; rather the move to other locations and communicate locally when they get there.

4 Foundational Issues

I conclude by listing some basic questions that have not been given a clear answer yet.

- What does a mobile computation do?

  This is the simple issue of meaning. It is has been typical to take an existing implementation model and extend it to the network with little regard for clean and consistent semantics. What are the consistent models mobile computation?

- Where does computation happen?

  To even ask the question of what mobile computations do, one must take the notion of multiple execution location as fundamental.

- How is mobile computation modeled?

  The notion of multiple execution location has been rarely considered in high-level models of computation. Where computation happens should have a visible influence (when appropriate) on behavior and resource usage.

- How do we reason?

  Locations are important because one must reason about the relative costs of computation and communication. Is it cheaper to perform a task here, or to move the code for the task elsewhere? Can one make this reasoning formal?

- What's the user model (language) for mobile computation?

  Many people have a ready answer: Actors, Network Objects, Threads, Closures, Continuations, Agents, etc. In fact, the user model should be tested against the unusual reality of Web programming, and is unlikely to turn out to be exactly any of the above.

- "But is it *really* secure?"

  The main obstacle to the acceptance of mobile computation for commercial applications is the issue of security. The basic technology is well known, but it is not clear how to deploy it into languages. What is the syntax, static checking, semantics, and logic of security?

References

[1] Cardelli, L., **A language with distributed scope**. *Computing Systems,* **8**(1), 27-59. MIT Press. 1995.

[2] Gosling, J., **Java**. Sun Microsystems. 1995.

[3] Ousterhout, J.K., **Tcl and the Tk toolkit**. Addison-Wesley. 1994.

[4] White, J.E., **Telescript technology: the foundation for the electronic marketplace**. White Paper. General Magic, Inc. 1994.