

# More than Patterns: Jini<sup>(tm)</sup> Technology and the Java<sup>(tm)</sup> Platform

**Jim Waldo**  
**Sun Microsystems, Inc.**  
**[jim.waldo@sun.com](mailto:jim.waldo@sun.com)**

The Jini technology, defined by Sun Microsystems<sup>1</sup>, is a distributed system technology designed to allow networks of services, composed of either hardware or software, to be assembled in an ad hoc fashion. The idea is that a service can dynamically join the network at any time, advertise the service that it wishes to offer, and be found by clients seeking that service.

There are certain aspects of the Jini technology that can be captured by a pattern language<sup>2</sup>. The way in which Jini services register or are found, the way those registrations (and other resources) are leased, the mechanism for registering interest in events and receiving notifications, and the notion of a transaction in a Jini federation are all capable of being captured in a pattern language. Indeed, many of the additional interfaces needed to define interactions between a client and a service, which are being defined in the Jini community, are also capable of being described in terms of a pattern language. Indeed, the very rules of the Jini development community have been stated in terms of such a language<sup>3</sup>.

The wide applicability of pattern languages to these fundamental constituents of the Jini technology have led some to the belief that Jini itself is just a set of patterns, and can be separated from the instantiation of those patterns in the Java language and environment. The reasoning behind such a belief seems to be something like the following:

- The Jini technology can be described in terms of a pattern language;
- Any pattern can be instantiated in multiple programming languages and environments;

therefore

- The Jini technology can be instantiated in multiple programming languages and environments

---

1. Arnold, O'Sullivan, Scheifler, Waldo, and Wollrath, *The Jini Specification*, 2000, Addison Wesley Press

2. as described, for example, in Gamma et. al., *Design Patterns*, Addison Wesley Press

3. Richard Gabriel and Ron Goldman, *Jini Community Pattern Language*, <http://jini.org/JiniCommunityPL.html>

While it is true that the Jini programming model can be used in different languages and in different environments by instantiating the patterns exemplified by the current Java implementation of Jini, it is not true that such an instantiation will give all of the abilities for ad-hoc networking that are present in the current Jini environment. Using the patterns (and therefore the programming model) may result in distributed systems that are more reliable and more scalable than those that use some other programming model, but they will not allow the same kind of spontaneous distributed computing model that is given by Jini.

Beyond the patterns that are an expression of the Jini programming model, Jini technology depends on the capability of having true distributed objects on the network. The client of a Jini service only knows the interface offered by the service. The actual implementation of that interface is dynamically downloaded as part of the lookup pattern for that service. Different services can offer different implementations, in the client address space, of the same interface, allowing Jini services to be independent of the wire protocol or data representation that is used for communication with the service. And this sort of functionality can only be offered in an environment like that provided by the Java platform.

To get this sort of functionality, the platform needs to support mobile objects, including both the data that makes up the state of those objects and the code that implements the methods for those objects. This mobility of the code used to implement an object cannot be supplied by adherence to any pattern language, but can only be provided by the underlying platform. In particular, the Java platform enables Jini by having the properties of

- portable object code (the Java bytecodes), allowing the implementation of an object to be moved from one node in the network to another;
- dynamic loading, which enables the objects moved to be run in existing, running Java virtual machines;
- ubiquity, allowing objects to be moved in the expectation that they will move to an environment in which they can be run;
- code safety and verification, which allows the recipient of mobile code to trust code enough to allow it to be run, and
- a polymorphic type system, that allows refinements of known types to be considered special cases of the types expected as parameters or return values in a distributed call.

Without these properties, the dynamic nature of the Jini environment would be severely limited, no matter what the programming patterns utilized were.

This does not mean that constructing a pattern language that describes the Jini programming model is not valuable. However, we should not be deluded into thinking that such a pattern language is all there is to Jini, or that the dynamic environment of Jini can be replicated by following such patterns.