Jini Community Pattern Language

Richard P. Gabriel & Ron Goldman

This document presents an excerpt from a pattern language being used to create and over time to evolve a community consisting of the various companies, universities, and individuals developing digital services based on Jini™ Connection Technology. The patterns shown are part of a ***sequence*** that would be used by anyone wishing to join the community and develop a new Jini service.

The Overall Context

The dominant architecture for connecting computers and electronic devices is currently to cluster dumb devices around a computer and then network the computers together. Advances in microelectronics now make it possible for devices to contain significant computing power. Recognizing this, Sun Microsystems developed Jini technology to facilitate communication between a world of intelligent devices. In this new model, devices offer their services—which are defined as classes that implement one or more abstract interface—to any applications, devices, or other services that wish to make use of them. This new architecture of digital services is highly distributed as opposed to the centralized nature of the current computer-centric model.

To fully realize the potential market for Jini technology requires Sun to work with a large number of partners. As a first step, Sun has made available the Jini source code under a modified Open Source license, called the Sun Community Source License or SCSL, and many thousands of individuals, universities and companies have downloaded the code. The Open Source model of software development was considered to be insufficient: A community of competitors turned cooperators around Jini technology needs to be developed to create the broad range of Jini services that are needed for a large, thriving marketplace to emerge, and Open Source was not designed for that.

What form this community should take is very unclear. This is a new way of doing business and there are no existing organizations that can be used as a model. The decentralized, non-hierarchical nature of Jini technology will need to be reflected in whatever organization the Jini community chooses.

Markets Versus Marketplace

In the following pattern language we make a distinction between the terms ***market*** and ***marketplace***. A *marketplace* is a collection of *markets* related by some commonality. In the built world, a marketplace is a often a square or street containing markets; in such a physical place, a market will often pop up if what it sells is likely to be bought by people who are buying things at the other markets. Therefore, there is a kind of affinity between the markets. It would be unlikely that a gourmet grocery store would be established in a row of heavy equipment sellers, but it might in an upscale mall with florists, gift shops, a pharmacy, and a fancy department store or two.

In our context, a Jini market might pop up for a small set of closely related devices or services. For example, Jini printers and Jini digital cameras might form a robust market. The Jini marketplace would comprise all the Jini markets; any pair of Jini markets might be linked only by the fact of their using Jini technology—for example, devices from different markets might not work together sensibly.

Normally, a company or even a trade association focuses on forming or growing a single market. What is unique about our approach is that we are trying to encourage the formation of the larger marketplace for Jini services by creating a community that will use this pattern language to work together and build itself.

Our Strategy

Our strategy is to work with members of the Jini community to develop a set of community principles that enable the community to self-organize and develop Jini services. These principles provide a framework within which to begin to design the community and its organization. Examples would be:

- Community activities should be *open*. Any community member should be able to follow electronic discussions and attend meetings.
- Community activities should be *fair* for all community members. We do not want an organization where all decisions are made by a few large companies, nor do we want one where a few small members can block all action.
- We need to *encourage diversity* so Jini services can range from experimental, innovative ones to those that are more mature and whose specifications are considered as standards.

Given the emerging set of principles, we combed development organizations, standards bodies, governments, political structures, and other successful human group endeavors for patterns which could be used to construct the Jini community and its subcommunities in such a way that the principles are embodied within the resulting structures.

The result is a pattern language, but with a little difference. Pattern languages have proven to be an excellent form in which to talk about how a set of structures and forces in a constructed artifact—a house, a city, a software system, or a development organization—combine to create an effective and livable instance of that artifact. In a pattern, you can talk about how a set of existing forces can be resolved in a particularly nice way to solve a design or construction problem. We have found that a pattern language can cause a set of abstract principles to emerge even though those principles are not explicit in the patterns.

The difference is that our pattern language does not construct only artifacts like ones seen before. Because the pattern language takes pieces and parts from successful organizations, it can generate not only variations on existing organizations, but entirely new ones as well. One example is the shepherded standards group. In this type of group, a technology expert is used to guide the work of domain experts to specify a new standard interface. Shepherds can then be organized into a panel which is policed by the community as a whole.

To further elaborate, some of the patterns in the pattern language talk about specific organizational entities. This does not mean that such entities are required to exist in the organization, but the pattern explains what forces would give rise to the need for such an entity. Thus, the pattern language is a sort of handbook for those creating the Jini Community to refer to while building and evolving the community, just as Christopher Alexander's book, **A Pattern Language**, is used every day by people working with architects and builders to make their homes or remodel them.

Use of Piecemeal Growth

The process we are using to create the Jini Community relies on the concept of ***piecemeal growth***. Rather than try to design the overall structure, we are working with smaller segments of the community as they emerge around common interests. For example, a group of companies that manufactures printers has coalesced, and we are working with them to create the structures they need to allow them to define Jini services.

As other groups form each will need to determine what organization is most appropriate for itself. Different groups will have different needs. Using the patterns in the language each group will generate the organizational structures best suited for the activities it needs to perform. Other patterns will help generate how these groups interact with the rest of the Jini Community.

In short, each company, individual, or group should use the pattern language to build its own part of the community, and the community as a whole should use the pattern language to adjust itself to newcomers. Each of the patterns in the pattern language that follows will be implemented by the companies, individuals, groups, or the community as a whole as they require solutions to the problems that the patterns address. And so it is by the myriad small acts of many people that the community and marketplace emerge.

Finally we expect both the Jini Community and the pattern language to evolve over time.

The Jini Community Pattern Language

The following is an excerpt from the Jini Community Pattern Language.

Summary of the Language

The patterns are presented so those addressing related concerns are grouped together. The current patterns fall into five rough areas: organizational process, the marketplace, fairness, product development, and quality and stability.

The first group focuses on some very general aspects about the process of organizing the Jini Community. The patterns in this group are:

- Gathering Requirements
- Evolve the Community
- Organization Follows Activities
- Adaptable Organization
- On-The-Fly Adaptation
- Organization Follows Geography
- Decision-Makers Near the Action
- Decision-Makers with Broad Oversight
- Go Whole Hog

The next group of patterns focus on how the organization will need to create a new marketplace and what the value added might be for companies.

- Dangerous Waterhole
- Safe Place to Share
- Grow the Market
- Encourage Diversity in Emerging Marketplaces
- Let the Market Decide
- Short Time to Market
- Coalitions Form Markets
- Strengthen the Brand
- Make the Most of the Least
- Drop Barriers
- The Heft of Quality
- Give Away the Least Expected
- Anchor Store

How to achieve fairness in the functioning of the Jini Community is the focus of the next set of patterns.

- Fair Processes
- Checks and Balances
- Protect the Weak
- Right to Appeal
- No Distinguished Members

- Appeals to Supermajority
- Information Flows Everywhere
- Ratification/Community Approval
- Proportional Votes
- Senatorial Votes

Next are several patterns concerned with various ways community members will develop products.

- Cut and Run
- Microcosm
- Make a Standard
- Single-Source Specification Creation
- Multiple-Source Specification Creation
- Compelling Idea

Finally there are the patterns concerned with building quality and stability into Jini services. These include a number of patterns that generate Open Source and Community Source development.

- Levels of Maturity
- Reward Stability
- Peers Ensure Quality
- Peer Review
- Community Ensures Completeness
- Shepherds Ensure Architecture
- Shepherd Panel
- Grow your Shepherds
- Shepherd Code of Ethics
- Community Reviews Shepherds
- Community Controls Development
- Compatibility Logo
- Give Away All You Can
- Every Bug is Trivial to Someone
- Survival of the Fittest
- Scratch Your Own Itch
- Work in Your Own Medium
- Continuous Releases

This is only a start on the patterns necessary to generate the organization needed for the Jini Community and to evolve it.

A Sequence to Create a Service

When a company, say, decides that it wishes to create a Jini service, that company first will see that it needs to approach or create a **Dangerous Waterhole** which will be occupied by its competitors. On closer inspection the company might determine that there are reasons to cooperate and that it is a **Safe Place to Share**. One of the most compelling reasons to cooperate is that creating a service alone might mean duplicating work that the community has already done and is continuing to do. Moreover, a service defined in a vacuum might find no real uses. By joining with the community's efforts, the company can concentrate on whatever value it can add, and it can **Make the Most of the Least**.

Next is the question of how the new service will be used. Is it a **Microcosm** only to be used internally by its creators or is it to be made available to the larger community? If the later, then will it be developed independently in a **Cut and Run** manner or in concert with other community members to **Make a Standard**?

The development of a Jini service involves the creation of specifications for interfaces and their correct implementation. To improve the quality of the resulting service, the company needs to decide whether to make use of peer review so that **Peers Ensure Quality** or to involve a community shepherd so that **Shepherds Ensure Architecture** (or both). In either case, the final service must pass the community specified compatibility testing if it is to receive the **Compatibility Logo**.

Finally, the service creator should **Give Away All You Can** by contributing as much of source code as possible back to the community in order to make the best use of community resources to further its development. When deciding this they should **Go Whole Hog**.

A Note on Form

The patterns in this sequence are in one of two forms. The skeletal form was used to quickly write down the pattern so we could look at it in the context of the pattern language. Later, as we grew comfortable with the pattern, we expanded it into an Alexandrian form.

Dangerous Waterhole

Context: A community is trying to create a new marketplace.

Problem: Competitors don't drink simultaneously for fear of attack.

Force: Competitors don't trust each other.

Force: There are no compelling reasons for the unformed marketplace to be able to be created nor for the result to have any value.

Force: No company can succeed alone: The marketplace—the range of possible markets—is too large.

Force: The community needs competitors both to demonstrate the existence and value of the potential marketplace and to fan out in a competition net to find the sweet spots. That is, you need natural selection to work for you.

Therefore: Make it so the waterhole is not too dangerous; make it safe for companies to cooperate; make the water cool and sweet. Recognize that competitors have secrets and want to protect them, but are willing to share secrets for a larger return. Know that competitors are predators and protect the marketplace from sabotage.

Remarks: Patterns that will make the waterhole less dangerous include: **Safe Place to Share**, **Give Away the Least Expected**, **Strengthen the Brand**, **Compatibility Logo**, **Fair Processes**, **Information Flows Everywhere**, and **Shepherd Code of Ethics**.

Known Uses: Open Source projects, industry consortia, NATO and UN peacekeeping groups, the courtroom, summer camp, the schoolroom, writers workshops, Visa International, and singles bars.

Safe Place to Share

Context: Any software development situation at a **Dangerous Waterhole** or a company or individual trying to **Make the Most of the Least**.

Problem: How can a community of competitor companies share those parts of the software that is crucial for the community's success while keeping private those parts on which competitiveness depends?

Force: There is a core of the software whose proper and effective operation is of value to all members of a potential community.

Force: Control of that core is not essential to any particular organization.

Force: Organizations do not want their competitors in control of that core.

Force: Intellectual property rights and/or processes have been already established or negotiated.

Force: Software is challenging—it requires careful, coordinated thought and planning.

Force: Software takes a lot of work—every development staff has much more work it wants to do than it can.

Force: Organizations do not want to share anything with their competition.

Therefore: Create a place or mechanism within which the software and its development can be shared, with the community in charge of future development. Each software-contributing organization needs to **Go Whole Hog** on giving up control.

Remarks: Patterns that will make it safe to share include: **Make the Most of the Least**, **Compatibility Logo**, **The Heft of Quality**, **Fair Processes**, and **Information Flows Everywhere**. This pattern directly resolves the "software takes a lot of work" force; more difficult is the "software is challenging" force, which can be resolved, perhaps, by this and other patterns that create a wealth of available programming gazorch that does not require as much planning and coordination which is solely required to minimize wasted effort and mistakes. In some sense, a development organization can make progress on a set of tough problems either by thinking and planning or by trying things out—we could say "by typing **`delete`** rapidly." In an Open–Source-like situation, there are more people to think and to type **`delete`**.

Known Uses: Open Source projects, Visa International, coffee clatches, the board room.

Make the Most of the Least

Context: A business in which effort or expense is required to earn money (for example, building automobiles rather than loaning money).

Problem: Fools and nice guys easily fail at business even when they have every good intention to please the customer.

Force: Many people believe, "if a little is good, more is better."

Force: Many people prefer to reinvent/reimplement things rather than use something created by an outside organization.

Force: Work is the wrong way to make money. "No one ever made money by typing."

Force: Maybe Tom Sawyer was right.

Therefore: Choose carefully the value you bring to the table, and figure out how to produce what remains for nothing or close to it.

Remarks: Safe Place to Share might be a way to complete this pattern.

Known Uses: Tom Sawyer, Microsoft, Linux.

Microcosm

Context: A member of the Jini Community developing a new service.

Problem: How can a member create a microcosm of services that talk privately to each other?

Force: The member believes there is a market for an interrelated set of services perhaps on a variety of devices.

Force: The member does not want to share the idea.

Force: The requirements specified by the Sun Community Source License (SCSL) on all community members to publish openly any specification intended to be shared with any third party.

Therefore: Create a private interface. Do not share it with a third party and implement it privately. Form any coalition around subcontractor relationships.

Known Uses: Drivetrains.

Cut and Run

Context: A member of the Jini Community developing a new service.

Problem: How does a community member move quickly with a product idea?

Force: A member has a great idea for a service.

Force: The member doesn't want to go through the hassle of a standards-like process or doesn't think it's necessary.

Force: The risk/reward equation appears to the member to favor being the first mover.

Force: The requirements specified by the SCSL on all community members.

Therefore: The member does the minimal amount of specification and testing code required by the SCSL and works the market alone or in conjunction with a couple of partners.

Known Uses: Normal business practice.

Make a Standard

Context: A member of the Jini Community developing a new service.

Problem: How does a community member pursue a product idea that requires some degree of standardization to succeed?

Force: A member or coalition of members decide that there might be a market for a new service.

Force: The member or coalition believes that the creation of the market requires a standards-like ratification process to help convince other companies to create complementary products.

Force: The requirements specified by the SCSL on all community members.

Therefore: The member or coalition engages the community process or creates one for ratifying specifications to some level of community standard.

Known Uses: Normal business practice, IETF, Visa International.

Peers Ensure Quality

Context: A community that is creating specifications for products in an emerging market.

Problem: How can the quality of specifications be guaranteed?

Force: No single member is able through its own expertise to make the highest quality specification.

Force: More eyes ensure the quality and coherence of a specification.

Force: The market is emerging, so that established standards are a long ways off.

Force: There are no external or objective measures of quality.

Therefore: Arrange for a system of peer review to ratify specifications. Make sure that the process of peer review includes steps that reduce the likelihood that good suggestions are ignored.

Remarks: Note that this solution can work alongside **Shepherds Ensure Architecture**.

Known Uses: Editorial review, writers workshops, IETF.

Shepherds Ensure Architecture

. . . in a community where specifications are part of the community working material necessary for the growth of the marketplace and where there is an overriding architectural integrity that must be maintained, there are situations where writing a specification requires knowledge both of the domain of the specification and of the architecture in which the specification operates.

* * *

Without experience writing specifications within a context where architectural integrity is important, it is easy to lose sight of the needs of the overall architecture. In a high-maturity market, the proposed specification requires deliberation because there are established players in the area; in a low-maturity market, there might be a desire to plan the structure of this part of the marketplace ahead of time.

Writing a specification should primarily be done by an expert in the domain that the specification covers. Yet, in the early stages of a new marketplace based on a new architecture, there will likely be no individuals or very few who are expert in both the domain of the specification and the new architecture.

Therefore,

Form a body of shepherds who are expert at the underlying architecture. Arrange the organization so that whenever a domain expert wishes to write a specification, a shepherd is assigned to make sure that the underlying architectural integrity is maintained.

* * *

Like all good shepherds—such as the editors of technical journals, PLoP shepherds, and built-world architects—the architectural shepherds should put the interests of the domain experts ahead of their own. See **Community Reviews Shepherds**, **Grow Your Shepherds**, **Peer Review**, **Peers Ensure Quality**, **Appeals to Supermajority**, and **Right to Appeal**.

Compatibility Logo

Context: A community needs to ensure compatibility among a set of independently developed products.

Problem: How can the community get a set of separate companies, usually competitors, to agree on compatibility.

Force: Competitors like to get an advantage over one another by making things different.

Force: The competitors would be better off being compatible, but there is no compelling argument in the form of existing forces.

Therefore: Create a logo that signifies to the customer the value of products in the new marketplace. Attach a compatibility requirement to license the logo. Charge a fee to pay for developing the logo (and building its value in the marketplace) and for any other development costs you need to bear.

Known Uses: Intel Inside, Underwriter's Laboratory, Visa International.

Give Away All You Can

Context: A development situation in a **Safe Place to Share**.

Problem: An organization has too much work to handle.

Force: Software is challenging—it requires careful, coordinated thought and planning.

Force: Software takes a lot of work—every development staff has much more work it wants to do than it can.

Force: Lots of the work to do is necessary, but does not add marketable value.

Force: Other organizations also need to do similar work that does not add marketable value.

Force: Lowering the barriers to entry might increase and/or mature the market more rapidly.

Force: People overvalue what they own and do not want to give it up.

Therefore: **Go Whole Hog** and give away as much of the work that does not add marketable value as possible. Turn over control of this work to the community, adding sweet water to the **Dangerous Waterhole**.

Remarks: This is a way to implement **Anchor Store**.

Known Uses: Netscape.

Go Whole Hog

Context: A world where organizations judge people.

Problem: People often know what is needed, but if it is a radical departure from normal practice they are reluctant to do it all and instead try to water it down to a small step in the right direction.

Force: The unknown is scary and hard to sell.

Force: People avoid change whenever possible.

Force: Being conventional is rarely punished.

Force: People want to cover their asses.

Force: You know what the right thing to do is.

Force: Half measures rarely work.

Therefore: Do the right thing to the maximum extent possible. Encourage others to do the same by supporting innovation and not penalizing failure.

Remarks: The world is full of trade-offs, but when a course of action is clearly superior we should take it even (and especially) if it makes us uncomfortable because it seems too radical.

Known Uses: Noyce and Sanders, founders of Fairchild Conductor in the 1960's, selling transistors at $1.05 each (the price of the competitive tube equivalent) when it cost $100 to make them.