Space-Based Programming

by Bernhard Angerer
03/19/2003

"Space-based programming" heralds a new way of building distributed applications. The dominant methods of distributed programming are based on remote procedure calls (RPC), most notably embodied in the technologies CORBA, EJB, and COM/DCOM. Space architecture supplies a surprisingly compact model that completely replaces the RPC paradigm. Its inherent, minimalistic approach predisposes it to a wide range of applications while endowing it with the advantages of modularity, scalability, and source code economy.

Yale University's Professor Gelernter took the first steps toward space architecture in the early 80s, when, in order to support distributed applications, he developed a programming language named Linda (see [Gel92]). Linda was composed of a small set of operations in combination with a globally-accessible persistent store, the so-called "tuple-space." These operations form a coordination protocol, orthogonal to classical programming languages and thus readily implemented within them. Research results have shown that a large number of problems in parallel and distributed computing can be neatly solved using this

architecture.

Distributed Shared Memory

By "space" we mean a virtual, distributed shared memory. The necessary API essentially contains four methods: `write`, `read`, `take`, and `notify`. `write` and `read` fetch and store objects in memory, respectively. `take` removes its object following a `read`, and `notify` announces changes to the space.

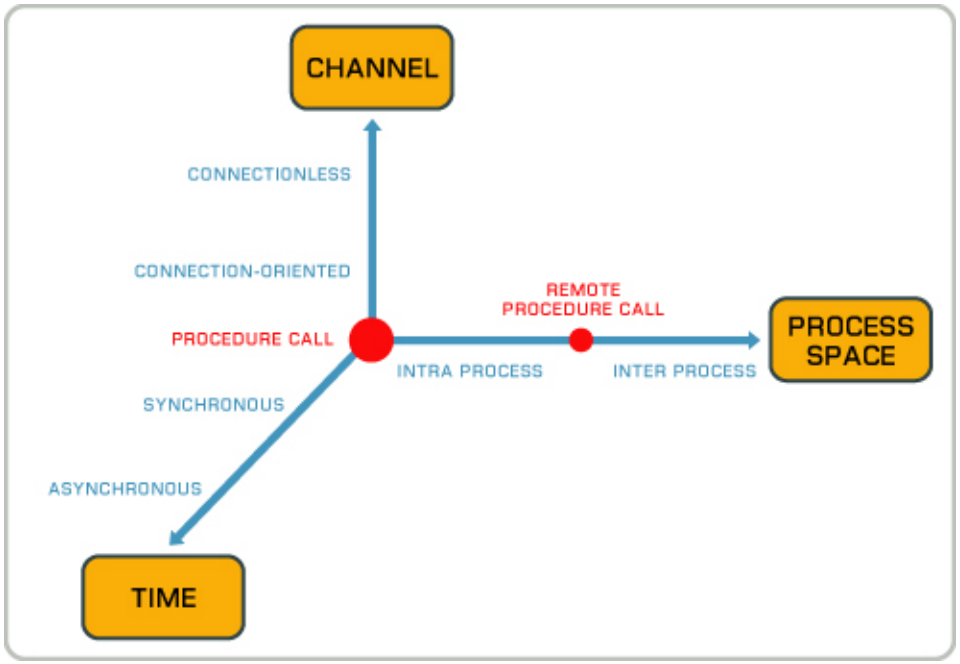

Figure 1: All degrees of freedom (see [Rup02])

One could not hope for a simpler conceptual schema of distributed systems. In an RPC model, objects communicate and synchronize their processes through explicit procedure calls. This approach doesn't scale particularly well, at least not without supplemental effort; in general, the greater the number of participating objects, the longer and the more convoluted their conversational paths become. In a space implementation, however, the space itself synchronizes all of its participants. This is "blackboard communication," in which the participants:

- Know nothing about one another (their exchange is connectionless and anonymous).
- Needn't share the same process or machine (interprocess).
- Are temporally independent of one another (asynchronous).

This radical disengagement in all three degrees of freedom (see Figure 1) leads to very good and flexible system design results.

The space or space service (compare [Fre99a]) provides the following services:

- **Simultaneous transparent access**: The space transparently satisfies the simultaneous data access operations of multiple processes distributed throughout a network. Distributed data structures are easily constructed, with the details of their communication remaining hidden from application developers. The actual data delivery mechanism (replication, propagation) functions behind the scenes and varies from implementation to implementation. Corso, for example (see the vendor/product list at the end of this article) provides two data replication methods: on-demand and pre-fetching. GigaSpaces, in its 2.x release, also provides replication mechanisms.
- **Persistence**: The space harbors a storage mechanism that guarantees that objects remain accessible until a process explicitly removes them. This permits, for example, the implementation of a chat application in which the participants needn't converse simultaneously. It is also possible to have the space discard an object upon the expiry of a preset lease.
- **Associativity**: Objects are located in space using associative lookups. The `read` method thus asks for an object by specifying the contents of its data fields, perhaps using null values as wild cards. The space search engine then uses value and type comparisons to return the desired object.
- **Transactions**: A transaction model enforces the atomicity of one or more operations hosted by one or more spaces.
- **Code Transfer**: Objects are present in space in serialized form; i.e., as passive data. Once read, however, an object's data fields can be modified or its methods called. In the case of Java, the class loader is used to bind raw objects to hitherto unused classes, thus endowing them with executable methods.

The simplicity of the fundamental paradigm is crucial; it can very easily be grasped, rendering unnecessary complex interfaces with their correspondingly steep learning curves. The minimal API allows for greatly reduced development expenditures; by dissociating the "senders" from the "receivers," it renders the application logic connecting them simpler, more flexible, and more stable. This particularly supports the construction, analysis, testing, and re-usability of large applications.

The Master-Worker Pattern

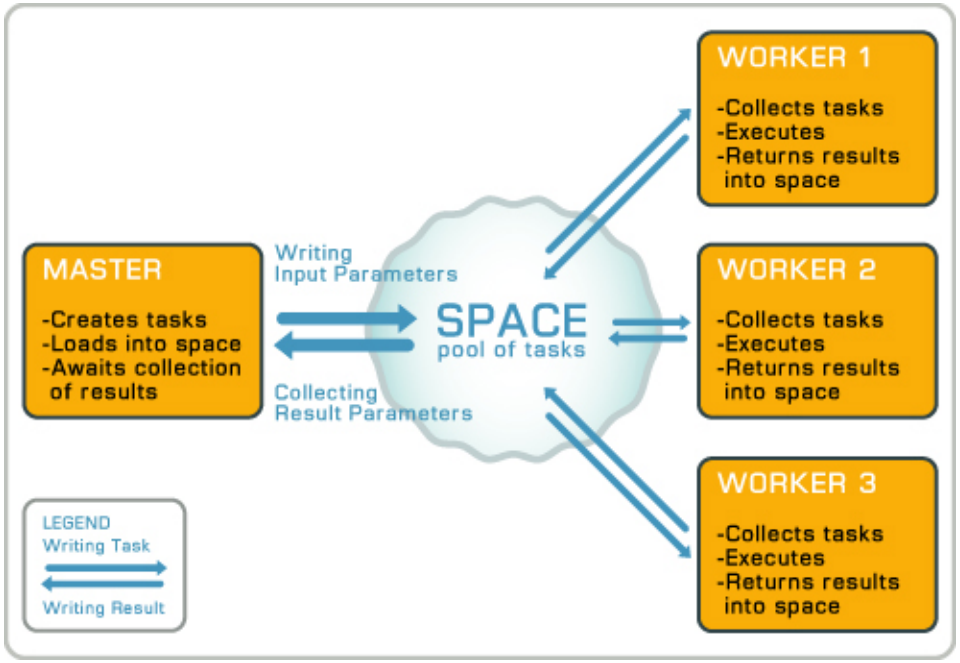

Figure 2: The Master Worker Pattern

The master-worker pattern nicely exemplifies the use of the space approach in developing a web application. Client requests (e.g., queries addressed to various archives) are received by a web server, which posts them asynchronously, as request objects, into the space (see Figure 2). The responsible worker processes then react in parallel, processing the requests and returning the results into the space as answer objects. The web server, upon notification of answers, then serves them to the clients, repackaging as required by the user interface.

By virtue of the use of a space as communication medium and the total decoupling of the interacting processes, this implementation scales linearly. In the event of a surge in the number of simultaneous requests, additional web server and worker machines (cheap standard hardware) can simply be added to the system as needed.

A worker represents an archive adapter. Should an archive be added, only its adapter need be newly programmed before being introduced into a running system. The author is aware of industry projects overjoyed at the benefit of this hot component plug-and-play capability.

Distributed Caching

Another characteristic almost inherent in spaces is distributed caching. Worker processes simply maintain a mapping of requests handled in a given timeframe to their resulting answers. Answer objects remain in the space, available to be returned immediately upon the arrival of a request identical to that of their mappings.

It is often difficult, by weighing costs and benefits, to achieve an optimal load balance in a system. Here again, a space offers a "natural" shortcut. Since workers are well-informed

about the load upon their respective machines, a "space proxy" is charged with request redistribution. Only requests for which a worker with sufficient resources can be found will be accepted. The load thus calibrates itself automatically amongst the worker processes and their machines without the need to distribute the information about the load itself.

The advantages of a space-based approach can thus be summarized:

- Faster development cycles.
- High scalability via the addition of cheap standard hardware.
- Shift of load from the web server to concurrent worker processes.
- Total decoupling of the interfaces.
- Distributed caching.
- Simple integration of heterogenous landscapes.

Project work in progress shows fantastic results with respect to the first point (compare [Sag01]). Bill Rawlings (CTO of Lockheed Martin, well known in industry circles due to his news postings) announced that by using spaces he was able to reduce the amount of source code (relative to J2EE) by two-thirds ([Raw01]). It's easy to imagine the effect on project budgets and timetables.

The following factors are essentially responsible:

- **Blackboard-based Architecture**: A space represents a new kind of message bus with a single interface over the entire heterogenous landscape. Processes are triggered and thereby synchronized.
- **Application-wide "Live" Objects**: For reasons of scalability, the database is often the only place where status information can be held permanently (stateless programming). With a space, the design can be greatly simplified, since both technical and business objects can bank their state in space. The impedance mismatch problem loses acuity.
- **Minimal API**: The simplicity of the space interface is the key to all of its salient characteristics. Interface descriptions (Ã la the CORBA IDL) are obviated--participants just follow their interface-programming paradigm, as they all reside in the same address space.

Vendors and Products

Sun has introduced JavaSpaces (see [Fre99b] and [Fle01]), based on Jini (see [Sin00]), which offers a rudimentary space as a reference implementation. The two vendors GigaSpaces and IntaMission follow this standard. TSpaces from IBM, though also implemented in Java, goes in a different direction, notably by aiming to incorporate database functionality. (The problem here is to provide a simple query language when the space must manage a very large number of objects.) TSpaces is not an official IBM product; it is in "research status" but can nonetheless be purchased. The Vienna-based Tecco has also gone its own way, in the process extending the original "tuple-space" model. In addition to associative searches, objects in space can be addressed directly, a novelty that enhances scalability and assists garbage collection. Corso runs "natively" under Windows, Unix, and Linux and offers language bindings for Java, C/C++, Visual Basic, and Oracle Forms; its runtime kernel is smaller than 1MB.

Summary

The beauty of the space architecture resides in its tandem of simplicity and power. Compared to other models for developing distributed applications, it offers simpler design, savings in development and debugging effort, and more robust results that are easier to maintain and integrate.

Considering the current marketing strategies of Sun & Co., it remains to be seen how long it will be until the J2EE community takes notice. Considering, on the other hand, the ever-increasing complexity of future scenarios (invoked, for example, by the keywords "mobile" or P2P) and presuming better communication between some companies' decision-makers and their developers, the situation might soon and radically change.

Literature & Links

[Gel92] D. Gelernter, *Mirrorworlds*, Oxford University Press, 1992

[Rup02] "Ruple White Paper - A Loosely Coupled Architecture Ideal for the Internet," RogueWave Software 2002

[Fre99a] E. Freeman and S. Hupfer, "Make room for JavaSpaces - Part 1," JavaWorld 1999, http://www.javaworld.com

[Sag01] D. Sag, "Jini as an Enterprise Solution," O'Reilly Network 2001

[Raw01] Bill Rawlings - Lockheed Martin, JavaSpace mailing list, Item 003081 oder 001588

Archives of JAVASPACES-USERS@JAVA.SUN.COM

[Fre99b] E. Freeman, S. Hupfer, K. Arnold, *JavaSpaces Principles, Patterns, and Practice,* Addison-Wesley 1999

[Fle01] R. Flenner, *Jini and Javaspaces Application Development,* Sams 2001

[Sin00] Sing Li, *Professional Jini,* Wrox Press 2000

Space Servers

- GigaSpaces/GigaSpaces 2.0, Herzelia, Israel
- IntaMission/IntaSpaces, Windsor, UK
- IBM / TSpaces, Almanaden Research Center, USA

Bernhard Angerer s a Software Architect focusing especially on object oriented methods and technologies.

Return to ONJava.com.

Comments on this article

1 to 3 of 3

Trackback from http://test.gleamynode.net/archives/000045.html
Space-based Programming
2004-02-11 07:59:36 [View]

difference to soap and wsdl ?
2003-07-09 07:56:46 anonymous2 [View]

There's an inherent problem with JavaSpaces
2003-03-20 06:57:25 anonymous2 [View]

There's an inherent problem with JavaSpaces
2003-04-02 12:27:53 anonymous2 [View]

There's an inherent problem with JavaSpaces
2003-03-20 18:05:49 tcopeland [View]

"Point to point", not publish/subscribe
2003-04-09 06:39:10 anonymous2 [View]

1 to 3 of 3