

T Spaces: The Next Wave

Tobin J. Lehman
IBM Almaden Research Center
toby@almaden.ibm.com

Stephen W. McLaughry
University of Oregon
stephen@cs.uoregon.edu

Peter Wyckoff
Bellcore
wyckoff@bellcore.com

Abstract

Millions of small heterogeneous computers are poised to spread into the infrastructure of our society. Though mostly inconspicuous today, disguised as nothing more than PIM (personal information management) computers, these tiny processors will eventually pervade most aspects of civilized life. The one thing holding them back from being everyone's portal to the new electronic society and the access point to an infinite store of information is the lack of a high-quality logical link to the world's network backbone.

Enter T Spaces, a network middleware package for the new age of ubiquitous computing. T Spaces is a tuplespace-based network communication buffer with database capabilities that enables communication between applications and devices in a network of heterogeneous computers and operating systems. With T Spaces, it is possible to connect all computers together, which leads the way towards an infinitely large cluster of cooperating machines. In this paper we describe the T Spaces package and explore some distributed applications that use T Spaces.

1 Introduction

The landscape of computing is shifting rapidly. Even as supercomputers are replaced by networks of personal computers, desktop PCs are being replaced by palmtop computers, which will soon be replaced, in turn, by embedded, wearable, and truly “ubiquitous” computers. This dramatic and rapid miniaturization of processors has left the software industry in a precarious position: As the number of different architectures and applications multiplies, the need for inter-operability between these systems becomes paramount. What is urgently needed is a high-quality logical link between the desktop, the palmtop and the tera-flop computer.

T Spaces, a project at the IBM Almaden Research Center, was created to address exactly this need. What is T Spaces? T Spaces is a combination of database, tuplespace, mobile computing and JavaTM. Most readers will be familiar with all of the above terms, except possibly for “tuplespace”. Tuplespace was popularized for parallel programming by the Linda project at Yale University back in the 1980's [ACG86, Ge185, CG89]. A Tuplespace is a globally shared, associatively addressed

memory space. The central concept of a Linda Tuplespace is very simple—agents communicate with other agents by writing, reading and consuming tuples (a tuple is an ordered set of typed values) in a universally visible Tuplespace. What makes the tuplespace model so attractive is that it replaces synchronous point to point communication with asynchronous (and anonymous) associatively addressed messaging, where the messages are kept in a persistent message store.

A tuplespace is a very robust and reliable mechanism for parallel and distributed applications. Since the communication is asynchronous and anonymous, a variable number of agents can work together on a task. And, since the message addressing is associative, a dynamic set of message receivers can register their interest and participate. In any direct addressing scheme, even with multi-cast, a truly *dynamic* set of message receivers would be hard to manage.

The new world of network-connected embedded computer devices, such as wireless 2-way PDAs (personal digital assistants), smart cell phones and automobile PCs is a perfect setting for the tuplespace communication model. Yet more than just tuplespace is needed to create the complete network middleware package. To connect all devices we must have a common language platform on which to run. The disparity in hardware platforms and language compilers can create a formidable barrier to creating seamless network middleware. Also, for information to flow freely between devices, network computers and mainframes, a significant cache of information must be managed by the middleware, which implies that some amount of database function is needed. Thus, a network middleware package that fits the upcoming requirements of the newly connected world uses the tuplespace communication model, a ubiquitous programming language and database support.

This paper describes T Spaces, the middleware component that represents the core of the ubiquitous computing project at the IBM Almaden Research Center. T Spaces combines database, Tuplespace and JavaTM technology to create a powerful new computing paradigm—a general platform-independent reposi-

tory that connects heterogeneous systems into one common computing platform. This combination of features represents a new interface model for a database system and, at the same time, the next generation of TupleSpace systems.

The structure of this paper is as follows. Section 2 discusses some of the previous work in TupleSpace systems, showing how it evolved from the original global communication systems to T Spaces. Given that there have been many variations on TupleSpace systems and few, if any, have had much of an impact on computing, we will also discuss why T Spaces is different from the rest. Section 3 presents an overview of the function of T Spaces. Note that we do not discuss the internal implementation of T Spaces, as that is covered in detail in the T Spaces architecture paper[35]. Section 4 shows some applications built on top of T Spaces, namely the network and mobile computing applications and the cluster of web crawlers. Finally, Section 5 concludes the paper.

2 Related Work

Today’s TupleSpace systems have their roots in the artificial intelligence blackboard systems that were created in the 1970s. Using the abstract notion of expert agents that could participate in a group computation, blackboard systems demonstrated the usefulness of a global communication buffer for collaborative computing. Later, the Linda TupleSpace systems and follow-on projects took a narrower view of the global communication buffer, using it almost exclusively for high-speed messaging between parallel processes of large high-performance computations. From there, several projects explored various combinations of Linda and database systems, Linda and JavaTM, and Linda and distributed systems.

2.1 TupleSpace: A Brief Description

A TupleSpace is a globally shared, associatively addressed memory space. The basic element of a TupleSpace system is a *tuple*, which is simply a vector of typed values, or *fields*. Tuples are associatively addressed via *matching* with other tuples. A TupleSpace is simply an unstructured collection of tuples. A tuple is created by a process and placed in the space via the *out* primitive (from the perspective of the client, the tuple is pushed *out*). Tuples are read or removed with *read* and *in* primitives, which take a tuple *template* and return the first matching tuple. (Note that, because the space is unstructured, the choice among multiple matching tuples is arbitrary and implementation-dependent.) Most

TupleSpace implementations provide both *blocking* and *non-blocking* versions of the tuple retrieval primitives. A blocking read, for example, waits until a matching tuple is found in the TupleSpace, while a non-blocking version will return a “*tuple not found*” value if no matching tuple is immediately available.

Using a very simple tupleSpace (the Colors tupleSpace in Figure 1) with single field tuples, we illustrate some of the tupleSpace operations. A **Client A** issued a *Read* command for a Color of type “Blue”. Since there was a tuple with that value, a copy of the tuple would be returned for either a blocking or nonblocking read. In the second case, **Client B** issued a blocking *In* command for a Color of type “White”. Since there are no Colors of that type, the client blocked on that command, waiting for a Color of that type to be added to the tupleSpace. In the third case, **Client C** issued a non-blocking *Read* command for a Color of type “Black”. In the non-blocking case, since there are no Colors of type “Black”, a Null value (the equivalent of a “tuple not found” value) is returned immediately.

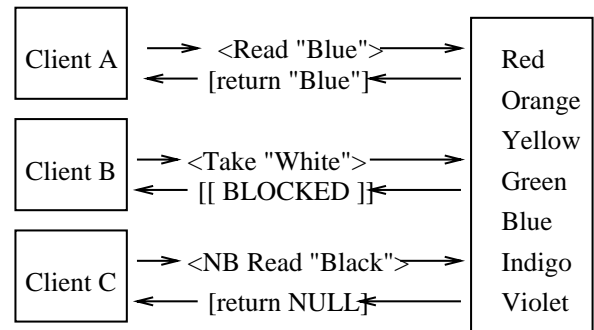


Figure 1: The Color Space Example.

TupleSpace provides a simple yet powerful mechanism for inter-process communication and synchronization, which is the crux of parallel and distributed programming. A process with data to share “generates” a tuple and places it into the TupleSpace. A process requiring data simply requests a tuple from the space. Although not quite as efficient as message-passing systems, TupleSpace programs are typically easier to write and maintain, for a number of reasons:

- Destination uncoupling (fully anonymous communication): Most message passing systems are partially anonymous: it is not necessary for the receiver of a message to identify the sender, but the sender always has to specify the receiver. The creator of a tuple, however, requires no knowledge about the future use of that tuple, or its destination.

- Space uncoupling: Since tuples are retrieved using an associative addressing scheme, multiple address-space-disjoint processes access tuples in the same way.
- Time uncoupling: Tuples have their own lifespan, independent of the processes that generated them, or any processes that may read them. This enables time-disjoint processes to communicate seamlessly.

Tuplespace extends message passing systems with a simple data repository that features associative addressing. Conceptually it ranks above a pure message passing system in terms of function, but far below relational database systems, since most implementations do not include transactions, persistence or any significant form of query facility.

2.2 Blackboard Systems

The idea of a globally accessible “data-space” for communication among multiple independent processes dates back to the *blackboard systems* used in Artificial Intelligence research since the early 1970s. Blackboard systems used the idea of a global slate (the blackboard) on which experts from various backgrounds could collaborate to solve difficult problems. The first and most famous of these systems was the Hearsay-II speech-understanding system developed at Carnegie Mellon University[13]. The system was structured as a group of diverse and independent agents called *knowledge sources*, which communicate through the blackboard (a global database). The knowledge sources search the blackboard for problem descriptions matching their domains of expertise, and write solutions for these problems (once they are calculated) back to the blackboard. The blackboard serves two purposes: representing intermediate states of the problem-solving activity, and communicating hypotheses about solutions among the knowledge sources.

The blackboard architecture was widely recognized as an interesting and flexible approach to the problem of integrating diverse independent agents into a cooperative system. However, until the early 1990s, few general-purpose blackboard systems were available, and each project that used this approach designed an ad-hoc solution[11, 3]. So even though the general idea of a collaborative space was a useful and constructive one, the global communication buffer didn’t catch on as a general distributed programming tool.

2.3 The Original Tuplespace

Although not mentioned in the literature by name until 1985, Tuplespace originated from a system developed at

SUNY Stony Brook in the late ’70s and early ’80s[17]. This system, the Stony Brook microcomputer Network (SBN), was a parallel computer composed of a set of processors in a torus-shaped communication grid. In order to support distributed computations on the network, a scheme was devised [5] to communicate the global state of the system to each node. The supported operations were *read* and *write*, which a node could use to find or update information about the network state. A systems programming language called Linda was designed for the SBN, consisting of two communication primitives, *in* and *out*. These were essentially wrappers for the read and write (respectively) operations provided by the network system. Linda also contained some novel parallel language constructs which later evolved into the study of “symmetric” programming languages but which will not be discussed here. As Linda evolved, a non-destructive read primitive was added (see [18]), and the concept of associative addressing by structured name was introduced. By 1985, the term “tuplespace” had replaced “global buffer” in the literature[19], and Gelernter had begun work on porting Linda to different architectures. The next year, Linda evolved into a set of communication primitives and C-Linda appeared in the literature (see [1]). Despite many changes in implementation and use, tuplespace has remained essentially unchanged since that time.

2.4 Tuplespace projects

For many years, Tuplespace research was restricted to the parallel programming community. Tuplespace systems were available in the form of commercial products and freely available research projects. The most famous tuplespace project, Linda, was trademarked by Scientific Computing Associates, and marketed to large financial enterprises as the simple solution to their parallel computing needs. However, there were many other tuplespace systems that were created for parallel computer. Since we describe them in a previous paper[34], we will simply list them here: POSYBL, Glenda, P4-Linda, The University of York Linda implementations, Javelin, Piranha, and Linder. With time, the tuplespace work expanded from pure parallel applications into the more diverse area of distributed computing. A number of projects, including Laura[30], JavaSpaces[32] and Jada[22] (which was part of Pagespace[27, 9, 10]), showed that decoupling clients and servers is easily accomplished using Tuplespace. Then, as Tuplespace implementations in C++ became more common, the opportunity to enhance the matching process became obvious. Object Space[28] and Objective Linda[25] added the basic ideas of object orientation to the standard

Linda model. At the same time, other researchers investigated combinations of tuplespace and database systems.

Although the similarities between Tuplespace and databases has long been recognized, surprisingly few projects have explored the connections between the two subjects. Persistent Linda (PLinda)[2] was the first project to add database functionality, including transactions and a simple query and join engine, to Linda. Perl-Linda did not extend Linda with database operations per se, but did combine Linda primitives and the Perl scripting language, creating a system for web developers who need a simple data repository[29]. Finally, the 1990s was also a period in which Linda became a target for fault tolerant systems, such as FT-Linda[4], MOM[6], and Persistent Linda 2.0 (PLinda2.0)[2, 24].

2.5 Discussion

Tuplespace was a hot research area in the 1980s, but since then interest has dwindled. Its rise and fall in popularity was strongly correlated with the rise and fall of research on parallel processing, and parallel hardware in particular. Although the Linda model for parallel processing is simple, and can be made reasonably efficient for applications where the amount of communication is much lower than the amount of computation, the basic work in the area was done thoroughly in the 1980s at Yale, and there has not been much commercial interest in such systems. Later work, such as the fault tolerance projects, used Linda as a testbed for their parallel and distributed ideas; they did not attempt to improve on or add functionality to the Linda model itself. Linda is a good communication mechanism and model for distributed and parallel research because of its flexibility and simplicity.

In the late 1980s, systems for building distributed applications cropped up. Linda is a good starting point for such systems because it provides communication, synchronization, and a simple data repository in one framework. Yet these systems did not attract much interest. Although they provided some heterogeneity, portability, and inter-operability, they never were able to provide a solution that addressed all the issues well.

Recently, Tuplespace has received renewed interest. This is the result of the platform independence and ubiquity of JavaTM, and the explosion of distributed applications on the World Wide Web. Linda is attractive for these systems because it is easy to implement and provides the tools needed to build simple distributed applications. These Web-based Linda systems are mostly targeted at distributed applications, but there is also work being done on parallel systems. The advent of

JavaTM and the growth in world wide connectivity has enabled distributed applications from brake systems in cars to super computers to inter-operate. There are thousands of different types of these devices and they all have varying communication needs; the challenge now is to provide them with a flexible communication substrate. We believe that a Tuplespace system, augmented with JavaTM and database technology, will go a long way towards meeting this challenge.

3 T Spaces

T Spaces is the marriage of tuplespace and database, implemented in JavaTM. The tuplespace component provides a flexible communication model. The database component adds stability, durability, advanced query capabilities and extensive data storage capacity. JavaTM gives it instant portability. Also, Java's ability to download new classes on the fly creates unparalleled flexibility in T Spaces' ability to perform new functions and handle new types.

3.1 Tuplespace Function Overview

In a sense, the main function of T Spaces is message processing. It follows then, that a client's view of T Spaces is that of a message center, or perhaps even a message database. In this message database, clients can create and delete tuple spaces, which house related families of message (tuples). For security reasons, clients are expected to log into the system with a user and password. The access permissions for any one user depend on which groups that user is associated with. A group has a specific set of permissions that define which operators that group can call on a particular space (each space has its own access control list).

There are two special spaces in a T Spaces server that users do not have general purpose access to – they are the galaxy space (the space of all spaces) and the administration space (where all of the users, groups, passwords and permissions are kept). These spaces are accessible only through specific commands (create space, delete space, define user, define password, etc).

Besides deciding on a space to put the tuples and having a user/password for the system, a user only needs to know how to call the T Spaces operators (described below) to use the T Space. In addition, if the standard set of operators is not enough, then T Spaces provides the ability for the user to define new operators dynamically – although, only administrators of a space are allowed to do this.

There are two more features in T Spaces that were added for the user's convenience. One is file support,

which works as follows. When a user has a large data object that would be inconvenient to read and load into a tuple as a single value, T Spaces allows the user to simply reference the file by URL (universal resource locator) in the tuple. The T Spaces client and server code cooperate and move this file transparently to the user. The other feature is transaction support. Although most users don't need it, there are times when it is important to group a set of T Spaces actions into one logical operation that either entirely succeeds (commits) or leaves no residual trace (aborts).

3.2 TupleSpace Operators

The basic T Spaces operators are a superset of the standard Linda primitives. *Read*, *Write*, and *Take* are provided, along with set-oriented operators *Scan* and *ConsumingScan*, and a novel rendezvous operator, *Rhonda*.

Write stores its tuple argument in a T Space. *Take* and *Read* each use a tuple template argument which is matched against the tuples in a space. (The T Spaces matching algorithms are described below.) *Take* removes and returns the first matching tuple in the space, while *Read* returns a copy of the matched tuple, leaving the space unchanged. If no match is found, *Take* and *Read* each return the JavaTM type null, and leave the space unchanged. Blocking versions of these are provided, *WaitToTake* and *WaitToRead*, which (if no match is found) block until a matching tuple is written by another process. (Linda programmers will recognize the semantics of these primitives as *out*, *inp*, *rdp*, *in* and *rd*, respectively.) *Scan* and *ConsumingScan* are analogous to *Read* and *Take*, but return the set of all matching tuples in the space. *Rhonda* takes two tuples, and exchanges them for matching tuples provided by some other process' *Rhonda* call.

In addition to this expanded set of built-in operators, T Spaces allows new operators to be defined dynamically. This enables programmers to enhance the power of T Spaces at runtime, providing a greater degree of flexibility and expressiveness than has previously been achieved with a tuplespace implementation. In T Spaces terminology, the implementation of an operator is a *handler*. T Spaces provides a *add_handler()* operator that allows the user to stretch the limits of T Spaces functionality.

The T Spaces matching algorithms are enhanced versions of the original tuplespace algorithm. In the standard case, the template is simply a tuple, with one or more formal fields. (A formal field has a type, but no associated value). A tuple matches the template when all of the following conditions hold:

1. The tuple and template have the same number of fields,
2. Each of the tuple's fields is an instance of the type of the corresponding field of the template, and
3. For each non-formal field of the template, the value of the field matches the value of the corresponding tuple field.

Condition (2) simply extends the Linda notion of exact type equivalence to an object-oriented notion of subtype equivalence.

Among several enhancements to this basic algorithm, the most interesting is the query on named fields. One of the differentiating features of T Spaces is that it builds an index on each named field in a tuple. This enables clients to request tuples based solely on values in fields with a given name, regardless of the structure of the rest of the tuple (or even the position of the named field within the tuple). For example, an index query of the form “(<foo> = 8)” will select all tuples (of any format) containing a field named <foo> with the integer value 8. It is also possible to specify a range of values to be found in the index.

More detailed examples of these and other tuplespace enhancements, including a description of the T Spaces event mechanism, can be found in the T Spaces architecture paper[35].

3.3 Database dependability

T Spaces employs a real data management layer, with functions similar to heavy-weight relational database systems, to manage its data. T Space operations are performed in a transactional context, which ensures the integrity of the data. All of the operations of a transaction are recorded in a log on stable storage before the transaction commits. In the event of a server failure, the log is used to replay the operations of committed transactions.

As mentioned in the previous section, the data manager indexes all tagged data for highly efficient retrieval. The expanded query capability provides applications with the tools to probe the data with detailed queries, while still maintaining a simple, easy-to-use interface.

The combination of tuplespace and database technology, implemented in JavaTM results in a system which addresses the communication and data-management needs of a wide variety of computing devices. This power and flexibility positions T Spaces as the cluster-computing solution for the next generation.

4 T Spaces Distributed Applications

In the early stages of the T Spaces project, we validated the design by building several distributed applications. These were a multi-user whiteboard application, a multi-user chat-room application and a distributed cut/paste buffer application. Although these applications were useful, easy to write and did demonstrate the power of the T Spaces programming model, they did not really stress the system in any way – nor did they show off many of the T Spaces features. As a result, it became clear that we needed to build much more ambitious applications that would stretch the T Spaces system and demonstrate its real capabilities.

Two main efforts were undertaken to build substantial infrastructure on top of T Spaces. The first effort was performed by a sibling group at the IBM Almaden Research Center, the Grand Central Station (GCS) project[15]. The GCS group used T Spaces for two types of communication duties: client data delivery and workstation coordinator for collaborative web crawling. The second effort is being performed by the T Spaces team. We are using T Spaces to build the infrastructure for a universal connection between lightweight/mobile computers and heavyweight network computing services.

4.1 T Spaces in Grand Central Station

The GCS team used T Spaces in two different parts of the system. First, T Spaces was the coordination mechanism for a team of collaborating web crawlers to intelligently crawl the digital universe. Second, T Spaces was the delivery mechanism that took answer data from the search engine and pushed it to various types of information clients. Figure 2 provides a visual overview of the design.

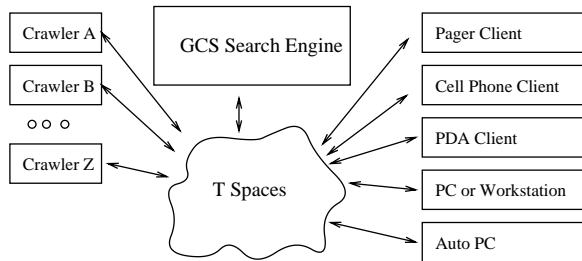


Figure 2: The architecture of GCS.

4.1.1 Collaborative Web Crawling: a High-Level Design

A collaborative web crawling (CWC) system uses multiple crawlers. These crawlers are designed to collaborate

in exploring a web-space, generating summaries, and storing these summaries for future reference. In order to achieve the maximum efficiency in CWC, these are coordinated so that the load is balanced and the overhead is minimized. Coordination is achieved by partitioning a web-space (the URL space) into sub-spaces and assigning each subspace to a processor. Each processor is responsible for building the summaries for those URLs that are contained within its assigned sub-space. During the construction of a summary, new URLs may be discovered by the Crawler. In this event, a process will keep processing those URLs belonging to its sub-space, including the new URLs if appropriate, route other URLs to proper processors.

To implement CWC, we use T Spaces for the communication between processors. The two types of information needed for the communication in CWC are: *foreign URLs*, where a processor needs to send a URL given by a crossing hyperlink to the processor assigned to it; and *coordination signals*, used to re-map and load re-balance the system,

Figure 3 shows a high-level description of our current CWC architecture.

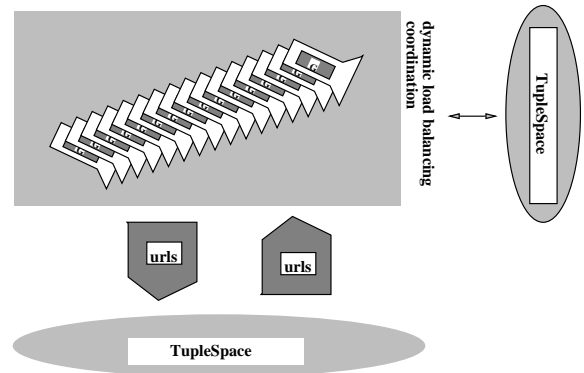


Figure 3: The architecture of CWC.

This has not yet been put into production, so we do not have any performance measurements of a fully active system. However, in a test environment, the foreign URL pool reached 20,000 URLs and the response time of the T Spaces server was fast enough so that it was not limiting to the team of crawlers. We plan on running more extensive tests in the future.

4.1.2 Information Delivery

As shown in Figure 2, GCS can support a variety of clients, so T Spaces must provide the delivery mechanism for a diverse set of devices, from fixed desktop

machines to smart mobile phones. These devices have different characteristics, such as speed, frequency of connect, capacity, and so on. Thus, T Spaces must be flexible enough to meet all of their requirements. For example, a desktop machine is typically available at all times, and thus can be notified with an event that more data is available. In this case T Spaces operates mostly as a passthrough buffer. On the other hand, a PDA or mobile phone may connect infrequently, so T Spaces must perform a several different duties. First, it must manage the data that it is holding for the client, possibly discarding some if the data exceeds the user's allocated space. Second, since the link from the mobile device to T Spaces is much slower, the user may want to selectively retrieve the data with a query, rather than download the entire set. Therefore, T Spaces must be able to randomly select and deliver (as a database system) the answer sets.

4.2 The Universal Information Appliance

Our initial experiences with the GCS information clients brought forth many useful experiences, including many custom built services and client applications. However, the majority of these involved the unidirectional flow of data from search engine to client, in the GCS case, and from clients to the various services. With our experience from GCS, our focus shifted to a more ambitious project involving general application support for mobile and disconnected clients. So, rather than simply deliver data in one direction, the universal information appliance (UIA) is the true two-way wirelessly connected PDA. Leveraging the T Spaces any-to-any promise, the UIA exchanges data, applications, and messages with other entities in the electronic universe.

The power of this model is readily apparent. By being connected to T Spaces, a device can contact any other device or service in the entire network. The decoupled Linda model offers persistent data and asynchronous communication to mobile and disconnected computers, making T Spaces the natural choice for the UIA. The architectural details of this system will be available in a future journal article.

4.3 Cluster Computing Discussion

The Berkeley NOW project [26] is a good example of today's cluster computing – a set of machines connected via a high-speed switched network. Its achievements (fastest web search engine, fastest disk-to-disk sort) are impressive – something not even dreamed about 20 years ago when mainframes still ruled the computing scene. Who could have imagined a collection of desktop machines beating a mainframe?

We predict that a similarly unimaginable scenario will involve tomorrows palmtop and embedded computers. Hooked together via high speed wireless links, a plethora of tiny processors will perform equally astounding feats of computation, though no doubt for some as yet undreamed of applications. T Spaces was created to work in this space – to connect all devices to all devices, but especially the new set of smaller devices. We've shown that T Spaces can work in today's workstation oriented setting. Our initial experience with collaborative crawling in GCS shows that T Spaces can operate as both a high speed communication buffer and as a database for storing tens of thousands of URLs.

The next step in computing will be somewhat abrupt. As the embedded devices evolve, and as the wireless infrastructure matures, there will be a day in the near future when a critical mass of embedded devices are connected to the network. On that day, the electronic climate will change, and full connectivity to the digital universe will be the norm. As history has shown, tomorrow's smaller computers are even more powerful than today's computers. So, just as today's PCs are more powerful than yesterday's mainframes, tomorrow's embedded devices will be more powerful than today's PCs. The future of cluster computing will not be the connection of tens or hundreds of PCs, but instead will be the connection and coordination of thousands or millions of heterogeneous devices, ranging from mainframes, to PCs to embedded computers. And, T Spaces will play a central role in connecting everything to everything.

5 Conclusion

The global communication buffer concept has had a long career, but not an overly prominent one. So far, all of the Blackboard systems and Tuplespace systems have existed in relative obscurity, even though the features they brought to the computing community were very useful. Our experience in building a large system and many small applications using T Spaces has shown us that this is a very powerful model and needs to be brought to the mainstream.

One reason it has stayed quiet is the availability. Most implementations have been university projects written in C or C++. As a result, they were not finished systems that could be deployed in the real world and they were not portable to many platforms. The real power of a system like T Spaces is having access to it from virtually every computer, from the microchip sewn into your clothing all the way up to the mainframe running the corporate (or federal) database.

A classical Tuplespace system alone, however, is not enough. Our experience has shown that even though

Tuplespace has an excellent communication model, it also needs a data repository and a more advanced query capability. This is the direction we've taken for T Spaces.

T Spaces seems to be a natural fit for the distributed world. It can provide the underlying network glue that ties all computing devices, big and small, together. Although it can improve the network connectivity in current environments (e.g. getting to that isolated Macintosh printer from a Windows 95 client), the real benefit will come when it is used to connect all mobile and embedded devices, making them all first class network computers, with full access to all network resources.

References

- [1] Sudhir Ahuja, Nicholas Carriero, and David Gelernter, *Linda and friends*, Computer, pp 26-34, August 1986.
- [2] Brian Anderson and Dennis Shasha, *Persistent Linda: Linda + transactions + query processing*, Workshop on Research Directions in High-Level Parallel Programming Languages, Mont Saint-Michel, France June 1991. Published as Springer-Verlag Lecture Notes in Computer Science 574.
- [3] Blackboard Technology Group, Inc. <http://www.bbtech.com>
- [4] David E. Bakken and Richard D. Schlichting, *Supporting fault-tolerant parallel programming in Linda*, IEEE Transactions on Parallel and Distributed Systems, 6(3), pp. 287-302, March 1995.
- [5] Arthur J. Bernstein and David Gelernter, *Storing and retrieving the network state: a survey and a proposal*, SUNY at Stony Brook Technical Report #80-011, October 1980.
- [6] S. Cannon and D. Dunn, *Adding fault-tolerant transaction processing to LINDA*, Journal of Software, Practice and Experience, 24(5):449-466, 1994.
- [7] Nicholas Carriero, Eric Freeman, David Gelernter, David Kaminsky, *Adaptive parallelism and Piranha*, Computer, 28(1), pp. 40-49, January 1995.
- [8] Nicholas Carriero and David Gelernter, *Linda in context*, Communications of the ACM, Vol. 32, No. 4, April 1989.
- [9] P. Ciancarini, A. Knoche, R. Tolksdorf, F. Vitali, *PageSpace: an architecture to coordinate distributed applications on the Web*, Computer Networks and ISDN Systems, Volume 28, Number 7-11, May 1996.
- [10] P. Ciancarini, A. Knoche, R. Tolksdorf, F. Vitali, *PageSpace: an architecture to coordinate distributed applications on the web*, <http://grunge.cs.tu-berlin.de:7500/www5/www350/overview.html>
- [11] Daniel D. Corkill, *Blackboard Systems*, AI Expert, Vol. 6, no. 9, pp40-47, September 1991.
- [12] A. Douglas, A. Wood, A. Rowstron, *Linda implementation revisited*, Transputer and Occam Developments, pp 125-138. IOS Press, 1995.
- [13] L. D. Erman, F. Hayes-Rogh, V. R. Lesser, D. R. Reddy, *The Hearsay-II speech-understanding system: integrating knowledge to resolve uncertainty*, ACM Computing Surveys, Vol. 12, No. 2, June 1980.
- [14] Kevin Eustice, Tobin Lehman, Armando Morales, "The Universal Information Appliance", IBM Systems Journal, 1999.
- [15] Daniel Ford, Qi Lu, Tobin Lehman, Matthias Eichstaedt, Peter Lazarus, John Thomas, *Grand Central Station: Information From Anywhere To Anywhere*, IBM Research Technical Report, 1997.
- [16] <http://www.fryselectronics.com>
- [17] David Gelernter, *An integrated microprocessor network for experiments in distributed programming*, SUNY at Stony Brook Technical Report #81-024, May 1981.
- [18] David Gelernter and Arthur J. Bernstein *Distributed communication via global buffer*, Proceedings of the ACM Symposium on Principles of Distributed Computing, pp. 10-18, August 1982.
- [19] David Gelernter, *Generative communication in Linda*, ACM TOPLAS, Vol. 7, No. 1, January 1985.
- [20] David Gelernter and Nicholas Carriero, *Coordination languages and their significance*, Communications of the ACM, Vol. 35, No. 2, February 1992.
- [21] Y. Gutfreund, J. Nicol, R. Sasnett, V. Phuah, *WWWinda: An Orchestration Service for WWW Browsers and Accessories*, 2nd Int. World Wide Web Conference, 1994.
- [22] <http://www.cs.unibo.it/rossi/jada/>

- [23] Keld K. Jensen and Gorm E. Riksted, *Linda, a Distributed Programming Paradigm*, Master's Thesis, Department of Mathematics & Computer Science, University of Aalborg, Denmark, June 1989.
- [24] K. Jeong, S. Talla, D. Shasha, P. Wyckoff, *An approach to fault tolerant parallel processing on intermittently idle, heterogeneous workstations*, Proceedings of The Twenty-Seventh International Symposium on Fault-Tolerant Computing (FTCS'97), pp. 11-20, IEEE, June 1997.
- [25] Thilo Kielmann, *Object-Oriented Distributed Programming with Objective Linda*, Proceeding of the First International Workshop on High Speed Networks and Open Distributed Platforms, June 1995.
- [26] <http://now.cs.berkeley.edu/>
- [27] <http://flp.cs.tu-berlin.de/pagespc/>
- [28] Andreas Polze, *Using the Object Space: A Distributed Parallel Make*, 4th IEEE Workshop on Future Trends of Distributed Computing Systems, pp. 234-239(6), Lisbon, September 1993.
- [29] W. Schoenfeldinger, *WWW Meets Linda: Linda for Global WWW-Based Transaction Processing Systems*, World Wide Web Journal, 1995.
- [30] Robert Tolksdorf, *Laura: A Coordination Language for Open Distributed Systems*, 13th IEEE International Conference on Distributed Computing Systems ICDCS 93 , pp 39-46, 1993.
- [31] Antony Rowstron and Alan Wood, *An efficient distributed tuple space implementation for networks of workstations*, Euro-Par'96, 1996.
- [32] SUN Microsystems, *JavaSpace Specification*, 1997.
<http://java.sun.com/products/javaspaces>
- [33] Shang-Hua Teng, Qi Lu, Matthias Eichstaedt, Daniel Ford, Toby Lehman, "Collaborative Web Crawling: Information Gathering/Processing over Internet," The 1999 Hawaii International Conference on Systems Sciences.
- [34] Peter Wyckoff, Stephen McLaughry, Tobin Lehman. "A History of TupleSpace Systems", IBM Research Technical Report. 1998
- [35] Peter Wyckoff, Stephen McLaughry, Tobin Lehman, Daniel Ford, "T Spaces", IBM Systems Journal, August 1998. pp 454-474