The Tuple Space:
An Old Solution to a New Problem?

Adrian Friday

Distributed Multimedia Research Group,
Department of Computing, Lancaster University, U.K.

E-mail: adrian@comp.lancs.ac.uk

1. Introduction

This paper discusses the impact on current distributed systems thinking of emerging and future networking technologies. In particular, we believe that future distributed systems platforms will be forced to abandon traditional thinking and embrace a new asynchronous model to justify their existence in the future. The world's internetworks are already beginning to be affected by the rapid development of two highly significant areas of research interest: mobile computing and high speed networking.

Mobile networks offer one of the most diverse and challenging environments in which to build distributed systems. A typical network is characterised by unpredictable connectivity, increased levels of bit error rate, end-to-end delay and jitter, dramatically reduced bandwidth and significantly, increased cost to the end-user. Furthermore, the diversity of bearer services offer applications little consistency in terms of service, for instance, style of interaction (connectionless versus connection-oriented), error protection and group support. Examples of such services include GSM, DECT and the emerging TETRA standard.

High speed fibre based networks (such as FDDI and ATM) are being deployed in response to the escalating demand for capacity driven by the rapid increase in the popularity of the internet. Future gigabit networks (currently under development) will offer transfer speeds in excess of many existing internal bus architectures [Boden,95]. The deployment of such networks in the near future will raise a great many issues. For instance, how to minimise the number of end-to-end delays incurred in interactions (latency reduction), how to utilise such a highly parallel network (distributed processing) and how to achieve scalability (particularly with respect to the predicted demand for more multimedia, e.g. video-on-demand/ conferencing applications).

As with mobile networks, in the future the popularity of consumer network devices (descendants of the emerging breed of network computer) will allow users to connect and disconnect to this global network at will. Furthermore, the availability of advanced integrated networking technologies such as these will enable the development of powerful distributed applications that facilitate collaboration, group-working and the sharing of multimedia data.

Experience gained during past research has shown that advanced applications (those incorporating aspects such as distribution, reliability, sharing of information and multimedia data) are particularly difficult to engineer. Such engineering tasks can be greatly simplified with the aid of the abstractions and management mechanisms provided by distributed systems platforms (for instance, the application of ISIS [Birman,90] to group management, or DASH [Anderson,90] to the management of continuous media).

However, in emerging advanced networks (those including mobile elements particularly), researchers are finding that the synchronous RPC paradigm, upon which most platforms are based, requires adjustment. For instance, to facilitate mobile working RPC messages are buffered (Mobile DCE [Schill,95]), delayed (MOST [Davies,94]) or queued (Rover [Joseph,95]).

Future networks will be characterised by thus far unparallelled network and system diversity. Multi-service mobile networks based on conventional, nano- and pico-cellular architectures will deliver a range of data connectivity options which will be combined with wired networked infrastructures to form a seamlessly integrated whole offering one or more potential networks at any given time. It is the contention of this paper that the synchronous RPC paradigm may be unsuitable for many applications in emerging and future networks.

In section 2, we examine a number of the implications of future networks for existing distributed systems paradigms. In section 3, the tuple space paradigm is highlighted as a suitable mechanism for addressing a number of these problems. In section 4, an analysis of the benefits of the asynchronous Tuple Space model is presented. Section 5 provides a brief overview of our initial work on a new distributed systems platform based on the tuple space paradigm. Finally, section 6 presents some concluding remarks and pointers for future work.

2. Existing Philosophies

One of the most common paradigms for constructing distributed applications is the remote procedure call (RPC). RPC provides a natural and semantically familiar mechanism to programmers by mimicking as closely as possible the local procedure call found in most programming languages. The success of RPC can be seen by its inclusion as a primitive in all prevalent distributed systems platforms (e.g. CORBA [OMG,91], DCE [OSF,91]) and as a prerequisite in interoperability models such as the RM-ODP [ISO,92].

The RPC is modelled on a local procedure call within a single process's address space, which by definition does not involve the network. It is considered unlikely that a local call will itself fail (although many languages offer exception based mechanisms to enable programmers to signal improper use of their code). In contrast, RPC requires that the caller (client) contacts the callee (server) and synchronises both in passing the arguments to the remote procedure and in returning any results.

RPC is thus based on the twin assumptions that, both client and server are available simultaneously and, that suitable actions can be taken to ensure the completion of the RPC should either the client or server fail. Since in real applications few operations are idempotent, providing RPC semantics in the presence of failure can be considered non-trivial. In conventional distributed systems platforms the conditions which constitute RPC failure can be considered sufficiently rare (e.g. network partitioning) that the paradigm remains largely successful.

However, as highlighted earlier, in modern networks comprising dial-up and mobile elements these important assumptions are often broken. In an attempt to allow conventional platforms to operate in these new environments, researchers have concentrated on providing new transparency mechanisms. For example, the work of Schill et al. [Schill,95] provides extensions to DCE to enable RPCs to be transparently rebound to alternative services (possibly local proxies based on cached data) in the event of network failure. In contrast, the MOST architecture [Davies,94] allows RPCs to be delayed, but in the event of failure provides mechanisms to enable applications to be informed, facilitating application-aware adaptation. The last example to consider is MIT's ROVER architecture [Joseph,95], which provides an object model in which RPCs are queued to a stable log in the event of failure (or disconnection) and are replayed when connectivity is restored. While the above solutions do, to some extent, hide the problems introduced by mobile networking, the mechanisms only function by breaking the implicit timeliness of the RPC.

In very high speed networks the massive bandwidth allows huge quantities of data to be shipped very quickly, however, the time taken for the first bit of information (regardless of the payload size) will always take the same time. Increases in performance can often only be found by optimising the number of end-to-end journeys (round trips). For instance, in a file system one might choose to ship an entire directory's contents when a user accesses a single file on the premise that (by the principle of locality) the user may wish to access the others also (thus reducing the number of round trip times taken to access the files). Since RPC mechanisms are based on the frequent exchange of (usually) small data payloads they are not well suited to latency critical applications in networks such as these (additional strategies involving server replication or proxies is required to achieve the necessary latency reduction).

In addition, the potentially unbounded delay in a given RPC requires that application programmers take evasive action to avoid their application's "locking up" in the face of network problems (e.g. event based models or heavy multithreading). This increased burden on the programmer is due directly to the synchronous paradigm implied by the RPC mechanism. As an alternative to RPC, in the next section we consider an asynchronous paradigm based on the concept of the tuple space [Gelernter,85a].

3. The Tuple Space Paradigm

The tuple space paradigm was conceived by researchers at Yale [Gelernter,85a] and has been extensively researched by the parallel programming community for over a decade. Tuples are typed data structures which consist of one or more typed data fields. Each data field is said to be an *actual*, if it contains a value, or a *formal*, if it does not. Collections of (possibly identical) tuples exist in a shared repository called a tuple space. Tuples can be dynamically deposited in and removed from the tuple space, but may not be altered "in-place". An amendment to a tuple is thus made by withdrawing it from the tuple space, altering its contents and reinserting it [Gelernter,85b]. Tuple spaces are shared between collections of processes, all of which have access to the tuples contained therein (illustrated in figure 1).

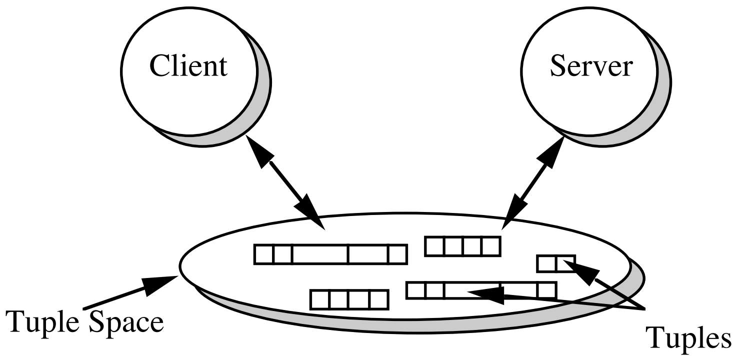

Figure 1 - Interaction via the Tuple Space

The tuple space paradigm was originally embodied in a coordination language called Linda. Linda consists of a set of tuple operators embedded in a host language (such as C) and does not exist as a standalone computational language in its own right.

The Linda operators perform four fundamental operations :-

i. `out` inserts a tuple, composed of an arbitrary mix of actual and formal fields, into a tuple space. This tuple becomes visible to all processes with access to that tuple space.

ii. `in` extracts a tuple from a tuple space, with its argument acting as the template (*anti-tuple*), against which to match. Actuals match tuple fields if they are of equal type and value; formals match if their field types are equal. If all corresponding fields of a tuple match the template the tuple is withdrawn and any actuals it contains are assigned to formals in the template. Tuples are matched non-deterministically and `in` operations block until a suitable tuple can be found.

iii. `rd` is syntactically and semantically equivalent to `in` except that a matched tuple is not withdrawn from the tuple space and hence remains visible to other processes.

iv. `eval` is similar to `out`, except it creates active rather than passive tuples. The tuple is active because separate processes are spawned to evaluate each of its fields. The tuple subsequently evolves into a passive tuple resident in the tuple space.

The implications of the tuple space paradigm for future networked environments is considered in the next section.

4. Applying the Tuple Space Paradigm to New Domains

The tuple space paradigm has a number of advantages which we believe suit it to operation in emerging and future networked environments. The tuple space model is an asynchronous paradigm which enforces no assumptions about timeliness, ordering or synchronisation. As long as tuples are unique (can be removed from the tuple space only once) the model continues to function. Since the matching of tuples to anti-tuples is non-deterministic, strict ordering does not need to be maintained which makes scalability easier to achieve. The interaction of processes is mediated through the tuple space (termed *generative communication*), decoupling the traditional client and server: any suitable server can generate a tuple of interest to any client (inter-process communication thus proceeds anonymously).

In a large network, operations with the tuple space will take a finite time to propagate to interested clients (this is acceptable since the matching of tuples is non-deterministic and does not imply fairness). Since physically nearer processes are likely to be able to respond (exchange tuples) more rapidly than remote processes, the model implicitly offers load balancing, improved fault tolerance, higher potential service availability and transparent service rebinding. Furthermore, since tuple based processes are often based on a producer/ consumer methodology (processes produce tuples when results are available which are consumed on-demand by client processes) the model may go part way to alleviate problems introduced by the network latency of very high speed networks (since the results may have already propagated to the client during time when the client is busy processing).

The space decoupling of the tuple space allows group interactions to be achieved transparently. A tuple produced by one process can be read by multiple clients in parallel (note that such applications need to be built around the *rd* primitive). Consider for example a groupware application like the shared whiteboard *wb* [Floyd,95]. The whiteboard process could output a tuple describing each drawing operation which is then read by the other whiteboard processes and rendered. Contrast this approach to existing RPC based paradigms which require group management protocols (or group bindings) to achieve a similar effect. Furthermore, new clients (whiteboards) can join and leave the tuple space at will.

A tuple is persistent, that is, resident in the tuple space until it is consumer (`in`'d). The persistence property of tuple spaces decouples processes in *time* as well as *space*. Unlike RPC, a client and a server are not synchronised by an interaction, the server may produce results as they become available and introduce them into the tuple space, the result tuples will remain (potentially after the demise of the server) until they are consumed by the client when needed at some (perhaps later) time. Processes can obviously only take advantage of the time/ space decoupling if they perform a function which can take advantage of the generative communication model. In the case where only a single server process performs a specific function based on client data, little advantage is gained by using the tuple space model.

A further advantage of the tuple space's persistence can be seen by once again considering the whiteboard example. Since drawing operations (tuples) are persistent, late entry into group interactions is possible since the entire group state is encapsulated by the tuple space. Explicit additional mechanisms would be required to manage the group state in a traditional RPC based application.

The tuple space can also be used to manage the state of applications. Taken to its logical conclusion, locally stateless applications (i.e. state entirely encapsulated by the tuple space) can be trivially migrated or (in some cases) replicated without the need for conventional checkpointing mechanisms. Furthermore, the tuple space can provide a medium through which agent based technologies can be implemented (via active tuples and the `eval` operation or the explicit exchange of mobile code in the tuples themselves).

5. Implementation

The viability of a platform based on tuple spaces as a distributed programming paradigm will hinge on the efficiency and scalability of the implementation in a distributed networked environment. Our initial implementation work is based on the Linda model, but includes a range of significant extensions which address the specific requirements necessary for operation in mobile environments. In particular, our system incorporates the following key extensions:

- multiple tuple spaces which may be specialised to meet application level requirements, e.g. for consistency, security or performance.
- an explicit tuple type hierarchy with support for dynamic sub-typing.
- tuples with QoS attributes.
- a number of system agents that provide services for QoS monitoring, the creation of new tuple spaces and the propagation of tuples between tuple-spaces.

The current implementation of our platform consists of a small stub library which is linked with each application process and a single daemon process, an instance of which executes on each participating host. Application requests (i.e. `in`, `out` and `rd` operations) are marshalled by the stub library and passed to the daemon process. The daemon process collaborates with other instances of itself on remote hosts to provide tuple space repositories and matching functions.

Instances of the daemon process communicate using a protocol called the Distributed Tuple Space (DTS) protocol. The DTS protocol maintains distributed local caches of tuples and requests (anti-tuples) and ensures that the caches reach eventual consistency.

To achieve scalability, it is essential that a distributed implementation does not apply locking strategies for operations which remove tuples and avoids use of algorithms which lead to acknowledgement implosion, both of which critically affect performance. Our protocol is based on IP multicast and borrows application level framing concepts from SRM the scalable multicast transport which underpins *wb* [Floyd,95] and *Jetfile* [Grönvall,96]. Essentially, each distributed tuple space is modelled as a multicast group (see figure 2).

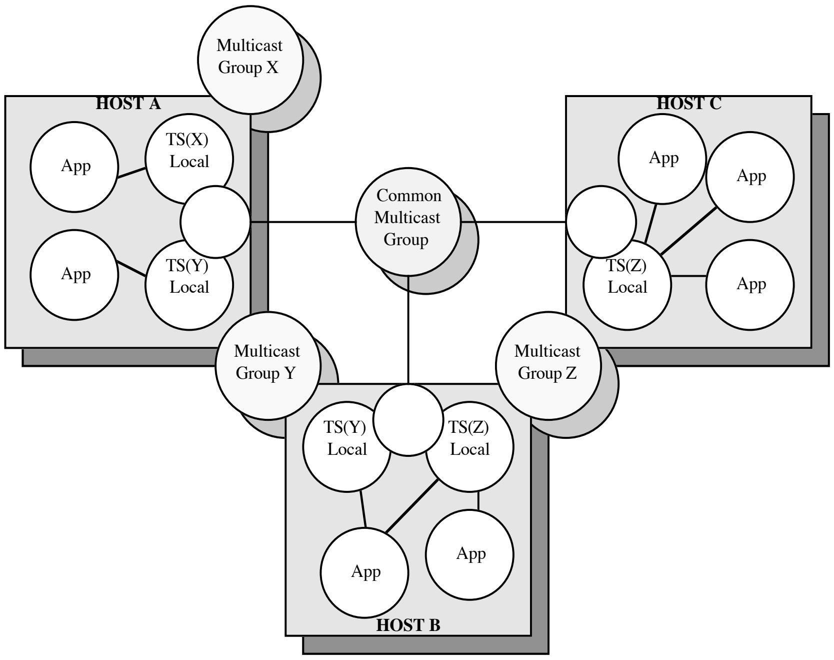

Figure 2 - Tuple Spaces are Modelled as Multicast Groups

Each tuple and anti-tuple is given a unique identifier which comprises a daemon identifier and a monotonically increasing integer. All messages in the protocol are multicast and it is assumed that the messages will be snooped by all available hosts in the multicast group.

The speed of the `in` operation governs the overall performance of the tuple space and causes the most problems in distributed implementations. The key to the performance of our prototype lies in the notion of tuple ownership. Importantly, ownership can be transferred to any of the participants of the tuple space and does not solely rest with the creator of the tuple. Before a tuple can be removed (by an `in` operation) the ownership of the tuple must be transferred to the performer of the `in` operation (achieved by nomination by the current owner of the tuple or by an explicit protocol message). Any user of the tuple space is free to `in` (remove) any tuples that they currently own without consulting any other user. Optimisations can be achieved through careful assignment of the tuple ownership.

We have built an implementation of the Limbo distributed systems platform which runs on Linux 2.0 (MULTICAST), SunOS 4.1.4 (MULTICAST-4.1.4) and Solaris 2.5. All these operating systems offer RFC 1112 compliant IPv4 multicast support, which is a prerequisite for running the DTS protocol. The platform consists of less than 6000 lines of C code with an executable size of 32117 bytes (Linux). More details of the Limbo platform can be found in [Davies,97].

6. Concluding Remarks and Future Work

In the future global internetworks will incorporate a far greater array of network services offering a huge range of communications options. The near future will witness a massive upturn in the number of mobile elements within the internet. Furthermore, the uses to which the networks will be put will escalate in complexity due to demand for multimedia data and advanced collaboration.

We believe current distributed systems paradigms (based on synchronous RPC) will not be suitable for these future advanced networks. As an alternative, we propose an asynchronous paradigm based on the tuple space coordination model. We believe the properties of the tuple space make it a far easier to manage the diversity and flexibility of these networks.

Early investigative work on an efficient and scalable platform based on the tuple space paradigm is underway. Initial results indicate that the platform is capable of performance of the same order of magnitude as many existing RPC based platforms (e.g. COOL [Chorus,96], ANSAware [APM,92]).

In the future we would like to investigate further how the platform will manage disconnection and reconnection (resynchronisation/ reintegration) to the network. In addition, we foresee that the platform will be required to operate over multiple integrated networks (perhaps many simultaneously) and the architecture for managing the transition between networks will require further study.

References

[Anderson,90] Anderson, D.P., S. Tzou, R. Wahbe, R. Govindan, and M. Andrews, "Support for Continuous Media in the Dash System." Proc. 10th International Conference on Distributed Computing Systems (ICDCS), Paris, France, May 1990.

[APM,92] APM Ltd., "An Introduction to ANSAware 4.0", Architecture Projects Management Ltd., Cambridge, U.K. February 1992.

[Birman,90] Birman, K.P., and R. Cooper. "The ISIS Project: Real experience with a fault tolerant programming system." Proc. ACM/SIGOPS European Workshop on Fault Tolerance Techniques in Operating Systems, Bologna, Italy, ACM Press, 3-5 September 1990.

[Boden,95] N. Boden, D. Cohen, R. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, Wen-King Su, "Myrinet: A Gigabit-per-Second Local Area Network", IEEE Micro, Vol. 15 No. 1, February 1995.

[Chorus,96] Chorus Systèmes, "CHORUS/COOL-ORB Programmer's Guide", Technical Report CS/TR-96-2.1, Chorus Systèmes, 1996.

[Davies,94] N. Davies, G. S. Blair, K. Cheverst and A. Friday, "Supporting Adaptive Services in a Heterogeneous Mobile Environment", Proceedings of the 1st Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, U.S., 8-9 December 1994, IEEE Computer Society Press, pp153-157.

[Davies,97] N. Davies, S. Wade, A. Friday and G. S. Blair, "Limbo: A Tuple Space Based Platform for Adaptive Mobile Applications", Proceedings of the International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97), Toronto, Canada, 27-30 May 1997. Internal report number MPG-97-02.

[Floyd,95] Floyd, S., V. Jacobson, S. McCanne, C. Liu, and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing." Proceedings ACM SIGCOMM, Cambridge, MA, ACM Press, Pages 342-356. August 1995.

[Gelernter,85a] D. Gelernter, "Generative Communication in Linda.", ACM Transactions on Programming Langauges and Systems, Vol. 7 No. 1, Pages 80-112. January 1985.

[Gelernter,85b] D. Gelernter, N. Carriero, S. Chandran and S. Chang, "Parallel Programming in Linda", Proceedings of the International Conference on Parallel Processing, pp 255-263, August 1985.

[ISO,92] ISO. "Draft Recommendation X.903: Basic Reference Model of Open Distributed Processing", Draft Report, ISO WG7 Commitee. November 1992.

[Joseph,95] A. D. Joseph, A. F. deLespinasse, J. A. Tauber, D. K. Gifford and M. F. Kaashoek, "Rover: A Toolkit for Mobile Information Access", Proceedings of the 15th Symposium on Operating Systems Principles (SOSP '95), Copper Mountain Resort, Colorado, U.S., 3-6 December 1995, ACM Press, pp156-171.

[OMG,91] The Object Management Group, "The Common Object Request Broker: Architecture and Specification (CORBA)", Technical Report 91.12.1, The Object Management Group. 1991.

[OSF,91] The Open Software Foundation, "Distributed Computing Environment: An Overview", Technical Report, Open Group Research Institute, 11 Cambridge Center, Cambridge, MA 02142. April 1991.

[Schill,95] A. Schill and S. Kümmel, "Design and Implementation of a Support Platform for Distributed Mobile Computing", Distributed Systems Engineering Journal, Volume 2, Number 3, 1995, pp128-141.