New and Improved: Linda in Java

George C. Wells

Department of Computer Science, Rhodes University,
Grahamstown, 6140, South Africa
G.Wells@ru.ac.za

Abstract

This paper discusses the current resurgence of interest in the Linda coordination language for parallel and distributed programming. Particularly in the Java field, there have been a number of developments over the past few years. These developments are summarised together with the advantages of using Linda for programming concurrent systems. Some problems with the basic Linda approach are also discussed and a novel solution to these is presented.

1 Introduction

Linda was proposed and developed in the mid-1980's by David Gelernter at Yale. There was a fair amount of interest in it as a model for parallel and distributed programming, but this waned through the early 1990's. In recent years there has been a considerable resurgence of interest in Linda, particularly in the Java community.

Linda is a language for distributed and parallel programming that has a very appealing simplicity. It is based on a simple shared-memory paradigm and has only a handful of operations. While this simplicity introduces other problems, particularly with regard to performance, these are not insurmountable and much research was done in the early days of Linda to develop techniques to ameliorate these drawbacks.

The first section of this paper presents a brief overview of Linda. This is followed by a survey of Java implementations of Linda, with an emphasis on the commercial developments in this area. Some of the problems that are inherent in the Linda model are then discussed, followed by the presentation of our solution.

2 Overview of Linda

Linda is a *coordination language* for parallel and distributed processing, providing a communication mechanism based on a logically-shared memory space called *tuple space*. Thus, the Linda model can be categorised as a form of *virtual shared memory*, in which the actual memory system may be physically shared or distributed, but application programmers are provided with a simple, shared-memory model.

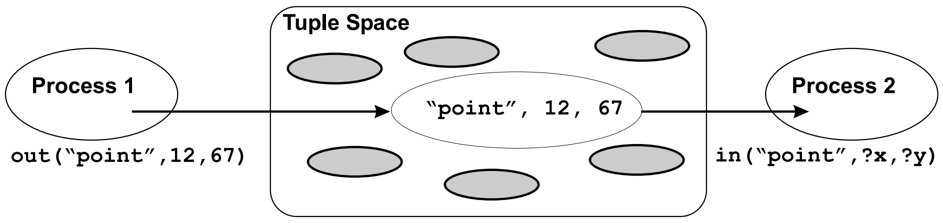

Figure 1: A Simple Communication Pattern

The tuple space is accessed using *associative addressing* to specify the required data objects, stored as *tuples*. An example of a tuple with three fields is `("point", 12, 67)`, where `12` and `67` are the x and y coordinates of the point represented by this tuple.

As a coordination language, Linda is designed to be coupled with a sequential programming language (called the *host language*—in our case, Java). Linda provides a programmer with a small set of operations. These operations may be categorised as *output* and *input* operations. There is a single output operation, used to place tuples into tuple space. This is called `out`, and is used as follows: `out("point", 12, 67)`. The input operations are used to retrieve tuples from tuple space. The basic forms are `in`, which removes the tuple from tuple space, and `rd`, which returns a copy of the tuple. The two input operations also have predicate forms (`inp` and `rdp`), which do not block if the required tuple is not present. The specification of the tuple to be retrieved makes use of an associative matching technique whereby a subset of the fields in the tuple have their values specified and these are used to locate a matching tuple in the tuple space. For example, the command `in("point", ?x, ?y)` might be used to retrieve the example tuple above (or any other tuple with a similar structure). The specification of the tuple used in an input operation is called an *antituple*. On successful completion of this input operation the variables `x` and `y` are bound to the values found in the matching tuple. The resultant communication between two processes is illustrated in Figure 1.

Further details of the Linda programming model may be found in [1].

3 Recent Linda Developments in Java

During the last few years a number of Linda implementations have been developed by research groups and commercial companies using Java as the host language. This paper considers the commercially-developed products, namely JavaSpaces, TSpaces, GigaSpaces and AutevoSpaces.

3.1 JavaSpaces

JavaSpaces[2] is a complex product and relies heavily on a number of other technologies developed by Sun Microsystems. As a result, configuring the JavaSpaces system and its applications is a very complex process.

JavaSpaces supports the basic Linda operations, although with slightly different names. Tuples (called *entries* in JavaSpaces) are created from classes that implement the Jini `Entry` interface, and only public fields that refer to objects are considered. Tuples are transmitted across the network using a non-standard form of serialisation. Matching of tuples is performed using byte-level comparisons of the data, not the conventional `equals()` method, and can make use of object-oriented polymorphism for matching sub-types of a class. Tuple storage is centralised, and this may become a performance bottleneck in large systems.

JavaSpaces provides some extended functionality, especially in areas such as support for *transactions* and *leases*, which are important for commercial applications.

3.2 GigaSpaces

GigaSpaces[3] was developed as a commercial implementation of the JavaSpaces specification. As such, it is compliant with the Sun specifications, while adding a number of new features. These include operations on multiple tuples, updating, deleting and counting tuples, and iterating over a set of tuples matching an antituple. There are also distributed implementations of the Java Collections `List`, `Set` and `Map` interfaces, and a message queuing mechanism.

Considerable attention has been paid to the efficient implementation of GigaSpaces. This includes the provision of facilities such as buffered writes, and indexing of tuples.

There is also support for non-Java clients to access GigaSpaces through the use of the SOAP protocol over HTTP. Lastly, there is support for web servers to make use of GigaSpaces to share session information.

3.3 AutevoSpaces

Like GigaSpaces, AutevoSpaces is a commercial implementation of the JavaSpaces specification. The focus of AutevoSpaces is on enterprise systems requiring high availability, including fail-over, recovery and load-balancing mechanisms. They claim that their "High Availability solution is the only commercially available implementation that provides semantic consistency with the JavaSpaces reference implementation. This consistency is essential to ensuring the correctness and flexibility of large, distributed, mission-critical applications"[4]. AutevoSpaces makes use of a distributed tuple space implementation to provide scalability and the high availability features.

At the time of writing, AutevoSpaces had only just shipped and so no practical evaluation of the system had been possible.

3.4 TSpaces

TSpaces is a Linda system developed by IBM's alphaWorks research division[10]. It is considerably extended from the original Linda model, particularly in terms of support for commercial applications. The TSpaces implementation is simple in comparison to JavaSpaces—all that is required is that a single server process be running on the network.

TSpaces supports a large number of operations, including new operations for the input and output of multiple tuples, and operations that specify tuples by means of a “tuple ID” rather than the usual associative matching mechanisms. There is also the `rhonda` operator, which performs an atomic synchronisation and data exchange operation between two processes. Lastly, there is an event mechanism providing notification when a specified tuple is written to the tuple space or deleted from it.

In addition to the usual associate matching technique, tuple input in TSpaces can be done using so-called “indexed tuples”. In this case, fields may be named, ranges of values may be used, and AND and OR operations may be specified. It is also possible to perform matching on XML data contained in tuples.

Tuples may have an expiration time set, and there is also transaction support. Furthermore, access control mechanisms are provided, similar to UNIX file permissions.

3.4.1 XMLSpaces

XMLSpaces is a research project, built on TSpaces to extend the limited facilities that it has for matching XML data[5]. The XML support in XMLSpaces is provided by subclassing the `Field` class used by TSpaces. The new `XMLDocField` class overrides the matching method used by TSpaces to provide matching on the basis of the XML content of the field. The matching method may be provided by the application programmer, providing a great deal of flexibility for XML matching operations. A number of matching operations are supported, including the use of XML query languages.

4 Problems with Linda

The simple associative matching mechanism used for the retrieval of tuples in Linda works very well in many situations. One-to-one and one-to-many communication patterns are trivial, and implementing semaphores, barrier synchronisation, and other coordination and interprocess communication models is simple. However, situations do arise where the simple associative matching technique is not adequate.

As a simple example, consider a set of tuples, where an application needs to locate the tuple with the minimum value of some field. Using Linda to solve this problem is possible, but is not efficient. The application would need to retrieve *all* of the tuples using repeated `inp` operations. These tuples would then need to be searched for the one with the minimum value. The tuples would then be returned to the tuple space (including the tuple with the minimum value, if the effect is to be that of a `rd` operation). During this procedure the tuples are not accessible by other processes, potentially restricting the degree of parallelism possible. Furthermore, in an implementation with a distributed tuple space, there is a large volume of network traffic generated by this solution.

While this is a simple example, it illustrates a general problem, namely that some applications may need a “global view” of the tuples in tuple space. Other examples include finding tuples with values “close to” some specified value, or lying within a specified range of values. These types of problems cannot be solved efficiently using the standard Linda associative matching technique. While some of the Linda systems described above, notably TSpaces, have provided extensions to the associative matching mechanism, none has addressed these issues.

5 The eLinda System

In an attempt to address the problem described in the preceding section, an alternative, flexible matching mechanism is proposed. We call this the *Programmable Matching Engine* (PME), and our Linda implementation *eLinda*.

The eLinda system is based closely on the standard Linda model. We have a number of implementations. The most complex of these uses a fully-distributed tuple space model where any tuple may reside on any processor/node. The others use a centralised tuple storage system, with optional local caching of certain tuples. The fully-distributed model poses particular problems for matching, in that many processing nodes may be required to participate in a matching operation. Further details of the eLinda system and its other features can be found in [6].

5.1 The Programmable Matching Engine

The Programmable Matching Engine allows the use of more flexible criteria for the associative addressing of tuples. This is useful in situations such as that exemplified above (finding the tuple with the minimum value for some field). As has already been noted, such queries can be expressed using the standard Linda associative matching methods, but will generally be quite inefficient. If the tuple space is distributed, searching for a tuple may involve accessing the sections held on all the processors. Ideally, this should be done in parallel. This problem is handled efficiently in eLinda by distributing the matching engine so that network traffic is minimised, and moving the necessary computation out of the application and into the matcher. For example, in searching for the minimum tuple, each section of the tuple space would be searched locally for the smallest tuple, which would then be returned to the matcher that originated the operation. The originating matcher would then select the smallest of all the replies received. This process is completely transparent to the application, which simply inputs a tuple, using a specialised matcher. From an application programmer's perspective this could be expressed simply as `in.minimum(?field1, ?=field2)`. The notation that is used is to follow the Linda input operation with the name of the matcher to be used[1]. The field (or fields) to be used by the matcher is denoted by `?=`.

In addition to this simple usage, matchers may also perform *aggregated operations* where a tuple is returned that in some way summarises or aggregates information from a number of tuples. For example, a matcher might calculate the total of numeric fields in some subset of the tuples in tuple space. It is also possible to write matchers that return multiple tuples, similar to the TSpaces "scan" operations.

New matchers are written as Java classes that implement a specific interface. This requires the implementation of two methods. One of these is used when checking all the tuples that are already in tuple space for a possible match. The other is used when the input operation has blocked and individual tuples need to be checked as they are added to the tuple space. These matching methods can make use of a simple

[1]Note that this is an idealised syntax, such as might be supported by a Linda preprocessor. In practice the usual style of Java method calls is used.

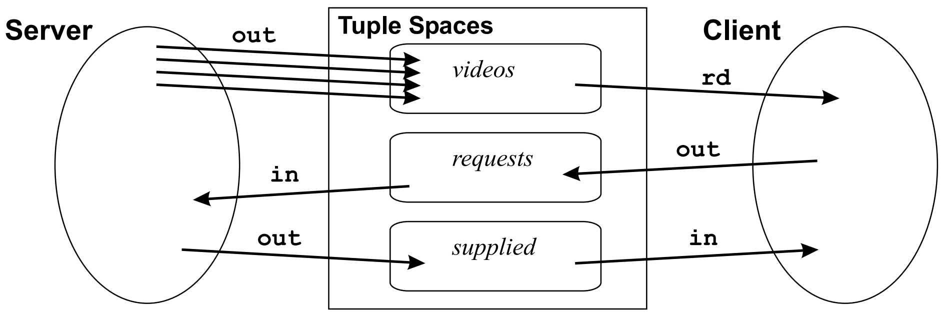

Figure 2: Control Flow in the Video Server Application

library that provides controlled access to tuple space and communication between the distributed matchers.

5.1.1 An Example Application: Video-on-Demand

As a simple illustration of the use of the PME (and also the multimedia support provided by eLinda) a demonstration video-on-demand system was developed. This consists of a server application that is used by the supplier of video resources, and a client application that is used by a customer wishing to view this material. A number of practical issues such as security, payment verification, etc. are omitted from this application for simplicity.

The Video Server Application This program initially places the details of the available videos into a tuple space called "videos". The server then waits for a tuple to be placed into a tuple space called "requests" with a matching supplier name. These request tuples specify a unique access key for the video required, and also contain payment details. The payment details are verified, and, if successful, a tuple is placed into a third tuple space, called "supplied". This tuple contains the unique key and a `MultiMediaResource` object that the client can retrieve in order to view the video. This process is shown diagrammatically in Figure 2.

The Video Client Application The outline of the client program is shown in Program Segment 1 (this has been expressed using a simple, procedural pseudocode notation for simplicity). This program is a GUI, event-driven Java application that allows a user to select a video and then view it. The user enters the title of a video, and the "videos" tuple space is then searched for a tuple with a matching title. This makes use of a PME matcher that retrieves the tuple with the minimum value in the cost field. Alternative matchers might also be provided for this purpose, which could take into account other issues, such as the quality of the video and the network capacity. If a matching tuple is found, the details are presented to the user and they are asked if they wish to view the video.

While this is a simple illustration of the principles involved in such an application, and particularly of the use of the PME and the multimedia features present in eLinda,

```
Get $videoName$ from user
if $videos$.rdp.$minimum$(?$supplier$, $videoName$, ?$key$, ?=$cost$) then
   Display video information
   if video is requested then
      $requests$.out($supplier$, $videoName$, $key$, $paymentDetails$)
      $supplied$.in($supplier$, $videoName$, $key$, ?$video$)
      $video$.play()
else
   Display “Video is not available”
```

Program Segment 1: The Video Client Application

it does provide a convincing demonstration of these facilities. In particular, it shows how the unique features of eLinda can simplify the development of such applications.

5.2 Limitations and Applications of the Programmable Matching Engine

There are some limitations to the kinds of matching operations that are supported by the PME. Notably, some matching operations may require a complete global view of the tuple space (e.g. where a tuple is required that has the *median* value of some field). In such situations the use of the PME may not be ideal, as all the tuples involved *must* be examined in order to find the result. However, it is important to note that such problems are handled no less efficiently than if the application were to handle them directly, using a conventional Linda system. Furthermore, the PME approach minimises the network traffic in such cases.

Most of the examples of matchers given above have been in the domain of numeric applications as these are simple to explain. However, the PME is not limited to numeric problems—it is just as applicable to textual, XML or other problem domains. One of the larger applications that was developed to test the PME concept is a graphical parser that utilised four specialised matchers. These performed tasks such as selecting a tuple where a field had a value that was a member of a specified set of values, and checking spatial relationships between coordinates stored as fields of tuples. Further details of this application of the PME are available in [9].

6 Results and Conclusions

Testing has shown that the performance of eLinda is on a par with that of other Java Linda systems[7], but that Java is currently not a viable platform for fine-grained parallel processing applications[8]. For coarser-grained distributed programming problems, the inherent simplicity of the Linda programming model is highly desirable and has led to the increasing interest in this approach, particularly in the Java community.

One of the weaknesses of the simple associative matching mechanism used in Linda is that it is limited for some applications. The Programmable Matching Engine developed for eLinda offers a solution that is itself simple and elegant, and caters for a range of different implementation strategies. The benefits of the PME have been confirmed through its use in a number of different application areas, with both distributed and centralised tuple space implementations.

Acknowledgments This work was supported by the Distributed Multimedia Centre of Excellence in the Department of Computer Science at Rhodes University, South Africa, with funding from Telkom SA, Comparex Africa, Letlapa Mobile Solutions and THRIP. The author also wishes to acknowledge the advice and support of Alan Chalmers (University of Bristol) and Peter Clayton (Rhodes University), and the helpful comments of the anonymous referees.

References

[1] N. Carriero and D. Gelernter. *How to Write Parallel Programs: A First Course*. The MIT Press, 1990.

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

[3] GigaSpaces Technologies Ltd. GigaSpaces. URL: `http://www.gigaspaces.com/-index.htm`, 2001.

[4] Intamission Ltd. AutevoSpaces: Product overview. URL: `http://-www.intamission.com/downloads/datasheets/AutevoSpaces-Overview.pdf`, 2003.

[5] R. Tolksdorf and D. Glaubitz. Coordinating web-based systems with documents in XMLSpaces. URL: `http://flp.cs.tu-berlin.de/˜tolk/xmlspaces/-webxmlspaces.pdf`, 2001.

[6] G.C. Wells. *A Programmable Matching Engine for Application Development in Linda*. PhD thesis, University of Bristol, U.K., 2001.

[7] G.C. Wells, A.G. Chalmers, and P.G. Clayton. Linda implementations in Java for concurrent systems. *Concurrency and Computation: Practice and Experience*. In press.

[8] G.C. Wells, A.G. Chalmers, and P.G. Clayton. A comparison of Linda implementations in Java. In P.H. Welch and A.W.P. Bakkers, editors, *Communicating Process Architectures 2000*, volume 58 of *Concurrent Systems Engineering Series*, pages 63–75. IOS Press, September 2000.

[9] G.C. Wells, A.G. Chalmers, and P.G. Clayton. Extending Linda to simplify application development. In H.R. Arabnia, editor, *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)*, pages 108–114. CSREA Press, June 2001.

[10] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. T Spaces. *IBM Systems Journal*, 37(3):454–474, 1998.