# Coordination Languages: Back to the Future with Linda

George Wells

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

**Abstract.** The original Linda model of coordination has always been attractive due primarily to its simplicity, but also due to the model's other strong features of orthogonality, and the spatial- and temporal-decoupling of concurrent processes. Recently there has been a resurgence of interest in the Linda coordination model, particularly in the Java community. We believe that the simplicity of this model still has much to offer, but that there are still challenges in overcoming the performance issues inherent in the Linda approach, and extending the range of applications to which it is suited. Our prior work has focused on mechanisms for generalising the input mechanisms in the Linda model, over a range of different implementation strategies. We believe that similar optimisations may be applicable to other aspects of the model, especially in the context of middleware support for components utilising web-services. The outcome of such improvements would be to provide a simple, but highly effective coordination language, that is applicable to a wide range of different application areas.

## 1 Introduction

This paper is based on more than a decade of experience with the Linda[1] model, and a number of projects, both developing and using Linda-like systems. This introductory section briefly describes the Linda programming model, outlines the history of Linda, and summarises our experience. The second section summarises some of the developments in this area in recent years. This is followed by a more detailed presentation of our eLinda system, and particularly the development of flexible matching mechanisms for input operations. This leads into a discussion of the open issues in this field.

### 1.1 The Linda Programming Model

The Linda programming model has a highly desirable simplicity for writing parallel or distributed applications. As a *coordination language* it is responsible solely for the coordination and communication requirements of an application,

[1] Linda is a registered trademark of Scientific Computing Associates.

relying on a *host language* (e.g. C, C#, or Java) for expressing the computational requirements of the application (this aspect is discussed in more detail in Section 1.2 below).

The Linda model comprises a conceptually shared memory store (called *tuple space*) in which data is stored as records with typed fields (called *tuples*). The tuple space is accessed using five simple operations[2]:

`out` Outputs a tuple from a process into the tuple space
`in` Removes a tuple from the tuple space and returns it to a process, blocking if a suitable tuple cannot be found
`rd` Returns a copy of a tuple from the tuple space to a process, blocking if a suitable tuple cannot be found
`inp` Non-blocking form of `in` — returns an indication of failure, rather than blocking if no suitable tuple can be found
`rdp` Non-blocking form of `rd`

The input operations specify the tuple to be retrieved from the tuple space using a form of *associative addressing* in which some of the fields in the tuple (called an *antituple*, or *template*, in this context) have their values defined. These are used to find a tuple with matching values for those fields. The remainder of the fields in the antituple are variables which are bound to the values in the retrieved tuple by the input operation (these fields are sometimes referred to as *wildcards*). In this way, information is transferred between two (or more) processes.

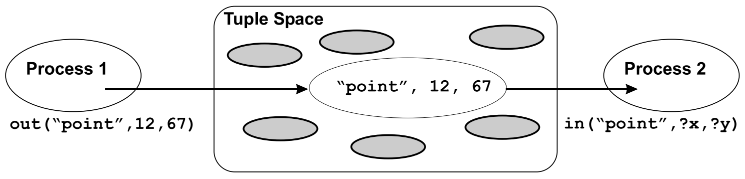

**Fig. 1.** A Simple One-to-One Communication Pattern

A simple one-to-one message communication between two processes can be expressed using a combination of `out` and `in` as shown in Fig. 1. In this case `("point", 12, 67)` is the tuple being deposited in the tuple space by Process 1. The antituple, `("point", ?x, ?y)`, consists of one defined field (i.e. `"point"`), which will be used to locate a matching tuple, and two wildcard fields, denoted by a leading `?`. The variables `x` and `y` will be bound to the values 12 and 67

[2] A sixth operation, `eval`, used to create an *active tuple*, was proposed in the original Linda model as a process creation mechanism, but can easily be synthesized from the other operations, with some support from the compiler and runtime system, and is not present in any of the commercial Java implementations of the Linda model.

respectively, when the input operation succeeds, as shown in the diagram. If more than one tuple in the tuple space is a match for an antituple, then any one of the matching tuples may be returned by the input operations.

Other forms of communication (such as one-to-many broadcast operations, many-to-one aggregation operations, etc.) and synchronization (e.g. semaphores, barrier synchronization operations, etc.) are easily synthesized from the five basic operations of the Linda model. Further details of the Linda programming model can be found in [1].

## 1.2 The Past

Linda was originally developed at Yale University by David Gelernter and his colleagues in the mid-1980's[2]. This novel approach to the coordination and communication between concurrent processes spawned many other research projects (see, for example, [3, 4]). The research at Yale also led to the establishment of a commercial company (Scientific Computing Associates) to develop and exploit these ideas[5].

**Coordination** Linda was the first coordination language, and the term *coordination language* appears to have been used for the first time in Gelernter and Carriero's 1992 paper "Coordination Languages and Their Significance"[6]:

> "We introduced this term [i.e. coordination language] to designate the linguistic embodiment of a coordination model. The issue is not mere nomenclature. Our intention is to identify Linda and systems in its class as complete languages *in their own rights*, not mere extensions to some existing base language. Complete language means a complete coordination language of course, the embodiment of a comprehensive coordination model."[6, p. 99].

In this paper they draw a distinction between the *computation model*, used to express the computational requirements of an algorithm (described in the previous section as the *host language*), and the *coordination model* used to express the communication and synchronisation requirements. They point out that these two aspects of a system's construction may be embodied in a single language (and much of their discussion is focused on refuting comments espousing this approach[7]), or may be embodied in two separate, specialised languages — their preferred approach.

**Advantages of Coordination Languages** This paper also gives a good exposition of what the originators of Linda perceived as the unique strengths of their approach. In essence, these are *orthogonality* and *generality.* They go to some lengths to defend the position that computation and coordination are orthogonal activities, and best supported by different languages. With respect to generality, they suggest that the concept of a general-purpose coordination language arises from the principle of orthogonality, and the comparison with general-purpose computation languages. This separation has advantages of portability (in the sense that a "Linda programmer" can adapt his/her knowledge of coordination and concurrent programming to new computation languages easily), and support for heterogeneity, which arises from portability:

> "we make it easier . . . to switch base languages, simplify the job of teaching parallelism, and allow implementation and tool-building investment to be focused on a single coordination model"[6, p. 101].

In the case of coordination, they define generality as the ability to cover the entire spectrum of concurrent activities: from multi-threaded applications executing on a single processor, through tightly-coupled, fine-grained parallel processing applications, to loosely-coupled, coarse-grained distributed applications. They cite the advantages of conceptual economy (or *simplicity*), flexibility and "intellectual focus" for the generality in a coordination language. In support of flexibility, they present a real example of a complex system, and then ask the following question:

> "Why should we accept *three* toolboxes, one for parallel applications (say, message passing), one for uniprocessor concurrency (for example, shared memory with locks), and one for trans-network communication (say, RPC), when *logically* Linda works well in all three cases?"[6, p. 104].

In summary then, their article is an impassioned plea for a simple, flexible, orthogonal approach to the construction of systems for a wide range of problems requiring concurrency at many different levels, and a presentation of Linda as a solution to this problem.

**Adaptive Parallelism and Reuse** Due to the temporal- and spatial-decoupling of communicating processes in the Linda model, it is an attractive platform for adaptive systems, where processing nodes may come and go during the lifetime of some system. The Yale Linda group did considerable work on this aspect, developing a Linda-based system for adaptive parallelism called Piranha[8, 9]. This was also produced as a commercial product by Scientific Computing Associates.

The decoupling of processes coordinated through a Linda tuple space also supports reuse, as a common tuple structure is all that is required to provide effective communication between components in a parallel or distributed processing system. If data-type conversions are handled by the Linda system, then the possibility of constructing heterogeneous processing systems becomes possible too.

### 1.3 Linda Today

After a great deal of initial enthusiasm for the Linda concurrency model, interest in this approach waned during the mid-1990's. However, in the late 1990's there was a resurgence of interest as a number of companies began to develop commercial implementations of Linda in Java. Among these was Sun Microsystems, which developed the JavaSpaces specification[10–12] as a component of the Jini system[13]. This specification has been adopted by a number of other companies, including GigaSpaces Technologies and Intamission, with their products, GigaSpaces[14] and Autevospaces[15], respectively. In addition, Scientific Computing Associates have developed a Java implementation of Linda called JParadise[16].

Sun's JavaSpaces specification provides for a Linda-like tuple space for data storage. There are a number of extensions present in JavaSpaces, in addition to the basic input and output operations, which have different names to the original Linda operations, but provide essentially the same functionality. These extensions are mainly focused on improving support for commercial applications, and include transaction support, leases for tuples (essentially a time-out, or expiration mechanism) and asynchronous event notification.

Independently, IBM developed a Linda system in Java, called TSpaces[17, 18]. This is similar to JavaSpaces in that it offers many of the same features for the support of commercial applications, but the implementation is considerably simpler and easier to use. The basic input/output operations have also been extended to include support for advanced matching (using named "index" fields, AND and OR operations), multiple-tuple operations and XML content.

In addition to these commercial developments, there are numerous recent and current research projects investigating various aspects of the Linda programming model, or using it as a platform for research in concurrent programming. A small selection of these projects may be found in the reference list[19–24].

### 1.4 Our Experience

Interest in the Linda model began at Rhodes University around 1990[25], with the local development of a Linda implementation, called Rhoda, for a Transputer cluster[26]. This was followed by the development of a platform for adaptive parallelism called Remora[27, 28], which was modeled on Yale's Piranha system.

In the mid-1990's, the author began the development of an extended Linda system, again targeting Transputer clusters, and with parallel rendering of photorealistic computer graphics as an application area[29, 30]. To support this research, an initial proof-of-concept system was developed using the Parallel Virtual Machine (PVM)[31]. Around 1997, it became apparent that the Transputer would no longer be developed or supported by the manufacturers, and a change of focus was required. Accordingly, the concepts that were embedded in the initial proposals were incorporated into a Linda-like system developed in Java[32–37] with additional support for multimedia applications. This system, called eLinda, was the central focus of the author's Ph.D. thesis[38]. The eLinda system is described in more detail below, in Section 2.

Recent work has also involved the use of the commercially-developed TSpaces system for a bioinformatics data-mining application[39, 40]. This research produced some very pleasing performance results, and quite coincidentally demonstrated some of the strengths alluded to by Gelernter and Carriero in the paper referred to in Section 1.2[6], such as the use of Linda for simplifying quite different aspects of the system (in this case, both distributed network communication and single-processor interprocess communication).

## 2 The eLinda Project

The initial goal of the eLinda research project was to investigate techniques for efficient communication in fully-distributed tuple space models (i.e. where any tuple can be stored on any processing node, with the possibility of duplication). This involved adding a new output operation to the Linda model, `wr`, which is intended for use with the `rd` input operation to suggest that broadcast communication is required. A secondary goal was to explore efficient support for multimedia communication, built upon the Java Media Framework (JMF)[41]. A later goal, which became the major focus of the project, was to generalise the associative matching technique used for the input operations in Linda. This led to the development of the Programmable Matching Engine (PME).

### 2.1 The Programmable Matching Engine

One of the weaknesses of the original Linda model is the simple associative matching technique that is used to locate suitable tuples for the input operations. This relies on exact matching (type and value) of any specified fields, and type-matching for the undefined fields (often called *wildcard*, or *formal* fields). This makes some simple input operations difficult to express efficiently. For example, if a tuple is required with the minimum value of a particular field, then the application must retrieve *all* matching fields (using `inp`), sort through these to find the one with the minimum value, and then return the remaining tuples to the tuple space. While this procedure is performed, other processes cannot access the tuples, potentially restricting the degree of parallelism that can be exploited.

The Programmable Matching Engine allows the programmer of an application to specify a custom matcher that is used internally during the retrieval of tuples from the tuple space. This can easily implement operations such as retrieving the minimum value. When the tuple space is fully-distributed, the PME allows the matching to be distributed. In the example above, the segments of tuple space held on each separate processor would be searched for the local minima and these returned to the processing node which originated the input operation. The matcher on this node is then responsible for selecting the global minimum from the resulting set of tuples.

There are still situations where a “global view” of the tuples in tuple space is required. A simple example of this is where the tuple with the *median* value of some field is required. In this case, a single matcher requires the values of the field in all the tuples in order to select the matching tuple. However, the PME still provides improved efficiency, as it is possible for a customised matcher to gather *only* the specified field values in order to determine the median value and then request this tuple from the processing node that holds it. These field values can also be communicated using a single network message. Both of these optimisations provide for more efficient network usage than if the application were to handle the problem explicitly without using the PME.

The development of new, customised matchers is supported by a simple library of support functions, providing developers with the ability to interact directly with the tuple space, and communicate with distributed components of the tuple space. The matchers themselves are written to conform to a simple Java interface, which requires only two methods to be written: the first to search the tuples currently in tuple space, and the second to be used when an input operation is blocked, as new tuples are added to the tuple space (this may simply do nothing if blocking input operations are not supported by the matcher).

### 2.2 Applications of eLinda

The benefits of the PME have been demonstrated by applying it to the problem of parsing graphical languages[42]. For this application four new matchers were developed. Two of these were general purpose, allowing for the retrieval of tuples where a field matches one of a set of possible values, and for retrieving a list of tuples that match some criterion, respectively. The other two were specific to the application, allowing for the input of tuples describing graphical components of the language that meet specified criteria (e.g. containment in a specified two-dimensional area). The Java classes implementing these matchers range in size from 67 to 130 lines of heavily-commented code, which, while not an accurate reflection of complexity, indicates the relative simplicity of using the PME.

Another application that was developed was a simple video-on-demand system[42]. A client program could search for a video, potentially retrieving a number of matching tuples from a number of suppliers. As implemented, a customised matcher allowed the retrieval of a video with the minimum value of the cost field. More complex matchers that took into account factors such as quality-of-service, available bandwidth, etc. could also be provided for such an application. The experience gained in developing this application suggested that the Linda model may be useful in supporting service-oriented architectures, and, more specifically, "web services" (see Section 3.1).

## 3 Open Issues

The underlying "open issue" that has motivated our previous research, and much of the other research on Linda, is the performance question: can a coordination model with a high level of abstraction provide reasonable efficiency for practical applications? Our experience, and the recent interest in Linda systems in Java, suggest that the Linda model is coming into favour, specifically as a mechanism for distributed applications running on general-purpose networks of workstations.

### 3.1 Linda for Heterogeneous Web Services

Our testing of eLinda, TSpaces, JavaSpaces and GigaSpaces revealed that the performance of these systems is not ideal for relatively fine-grained parallel-processing applications, such as parallel ray-tracing[33, 36]. Focusing on distributed applications, and the current interest in web-services (or, more generally, *service-oriented architectures*) for distributed processing, suggests that the implementation of the Linda model as middleware for web-services would be a useful avenue of exploration. The ease-of-use of the Linda model and the spatial- and temporal-decoupling that it provides would be extremely beneficial as a middleware layer for applications based on web services. This appears to be largely unexplored territory at present, except for the work of Lucchi and Zavattaro[23], who focus specifically on the security of a tuple space web-service.

Our intention is to reimplement the eLinda system as an XML-based web service. This raises a number of questions, as yet unanswered, as to the best approach to take. The web-services community appears to be divided between the use of RPC models, based on protocols and standards such as SOAP and WSDL, and the direct use of XML and the basic World Wide Web protocols and standards (an approach referred to as Representational State Transfer, or REST)[43, 44]. The relative simplicity of the REST approach is appealing, but there are questions around issues such as security and reliability, which require investigation before implementation commences. The opportunity will also be taken to redesign some of the fundamental features of the eLinda system, which should also produce some general performance improvements.

As a central issue in redeveloping the eLinda system as middleware for web services, we plan to investigate language interoperability issues, specifically comparing the use of C# and Java for client and server implementations, and heterogeneous system configurations, but not limited to these two languages. This will support an investigation of the interoperability issues between applications developed in different programming languages making use of a common Linda tuple space service, and will also permit comparative performance studies. Interoperability of data-types expressed in XML is a thorny issue and we expect to face considerable difficulty in this regard. However, we believe that the Linda web-service approach has great promise for supporting component reuse and simplifying the development of complex systems composed of web-services. In particular, Linda's spatial decoupling provides a platform for the simplification of problems such as service discovery.

If the language interoperability problems can be solved, then we will potentially have a very useful mechanism in place to allow for heterogeneous system components to be combined in very flexible ways. The strongly-decoupled nature of Linda may be particularly useful for solving problems involved in adaptation. It is hoped that this aspect can be explored further during the workshop discussions.

### 3.2 Improved Flexibility

Comparison of the Programmable Matching Engine with related features of other extended Linda dialects (particularly TSpaces[17, 45] and I-Tuples[46]) suggests that there may be some benefit to be had from applying similar techniques to *output* or *update* operations[38]. The extended update operations are intended to optimise the common sequence of retrieving a tuple, modifying a field's value, then returning the tuple to tuple space. For simple operations, such as incrementing a numeric field, there may be considerable efficiency gains to be had if the operation is carried out by the server, directly in tuple space, minimising the network communication required. Extended output operations are typically quite simple (e.g. TSpaces' `multiWrite` command), but may provide useful reductions in the load imposed on the network.

The Programmable Matching Engine in eLinda was developed specifically as a mechanism for increasing the flexibility of the matching process used for *input* operations. Despite this, it can be used, albeit awkwardly, to emulate the update and extended output operations of these other Linda systems. Providing a better design for the handling of output or update operations would be a useful extension to the demonstrated benefits provided by the PME.

Our plan is to implement new update operations analogous to the existing flexible input operations provided by the eLinda PME. This will be followed by application development and testing to assess the benefits of these new operations. Extended, flexible output operations will also be considered, but these should be relatively simple to implement and test. These extensions will be compared, both quantitatively and qualitatively, with the existing commercial and research systems that have adopted similar extensions. Formal modeling of the PME and these new extensions would also be extremely desirable. As always, the goal will be to assess whether the enhancements can improve the flexibility and performance of Linda systems, while preserving the fundamental simplicity of the model.

---

The wide-spread commercial development of Linda implementations in Java indicates that there is still much interest in the original Linda model of coordination. However, research (our own, and that of others) also indicates that there is considerable scope for improving both the performance and flexibility of use of the Linda programming model. While our prior research has addressed some specific issues in these areas, much remains to be done.

In summary, our hypothesis is that there is considerable scope for the adoption of the original Linda coordination model, with some extensions, as a simple, flexible coordination mechanism for distributed applications running on general-purpose networks of workstations. In particular, we envisage a place for Linda as middleware for heterogeneous, spatially-decoupled components executing in a web-services environment. This hypothesis needs to be tested by in-depth quantitative and qualitative investigation of the implementation and use of a Linda system in such an environment.

**Acknowledgments**

The eLinda research project is supported by the National Research Foundation (NRF) of South Africa, and by the Distributed Multimedia Centre of Excellence in the Department of Computer Science at Rhodes University, with funding from Telkom SA, Business Connexion, Comverse, Verso Technologies and THRIP. The author particularly wishes to thank Alan Chalmers of the University of Bristol, and Peter Clayton for their valuable advice and support over many years, and Peter Wentworth for the many thought-provoking discussions on these topics.

## References

1. Carriero, N., Gelernter, D.: How to Write Parallel Programs: A First Course. The MIT Press (1990)
2. Gelernter, D.: Generative communication in Linda. ACM Trans. Program. Lang. Syst. **7** (1985) 80–112
3. Banâtre, J., Métayer, D.L., eds.: Research Directions in High-Level Parallel Programming Languages. Volume 574 of Lecture Notes in Computer Science. Springer-Verlag (1992)
4. Wilson, G.: Linda-like systems and their implementation. Technical Report 91-13, Edinburgh Parallel Computing Centre (1991)
5. Scientific Computing Associates: Home page. URL: `http://-www.lindaspaces.com/index.html` (2004)
6. Gelernter, D., Carriero, N.: Coordination languages and their significance. Comm. ACM **35** (1992) 97–107
7. Kahn, K., Miller, M.: Technical correspondence. Comm. ACM **32** (1989) 1253–1255
8. Carriero, N., Gelernter, D., Kaminsky, D., Westbrook, J.: Adaptive parallelism with Piranha. Technical Report 954, Yale University (1993)
9. Kaminsky, D.: Adaptive Parallelism with Piranha. PhD thesis, Yale University (1994)
10. Freeman, E., Hupfer, S., Arnold, K.: JavaSpaces Principles, Patterns, and Practice. Addison-Wesley (1999)
11. Sun Microsystems: JavaSpaces service specification. (URL: `http://-java.sun.com/products/jini/2.0/doc/specs/html/jsTOC.html`)
12. Bishop, P., Warren, N.: JavaSpaces in Practice. Addison Wesley (2002)
13. Sun Microsystems: Jini connection technology. (URL: `http://www.sun.com/jini`)
14. GigaSpaces Technologies Ltd.: GigaSpaces. URL: `http://www.gigaspaces.com/-index.htm` (2001)
15. Intamission Ltd.: AutevoSpaces: Product overview. URL: `http://-www.intamission.com/downloads/datasheets/AutevoSpaces-Overview.pdf` (2003)
16. Scientific Computing Associates: Virtual shared memory and the Paradise system for distributed computing. Technical report, Scientific Computing Associates (1999)
17. Wyckoff, P., McLaughry, S., Lehman, T., Ford, D.: T Spaces. IBM Systems Journal **37** (1998) 454–474
18. IBM: TSpaces. (URL: `http://www.almaden.ibm.com/cs/TSpaces/index.html`)

19. De Nicola, R., Ferrari, G., Meredith, G., eds.: Proc. 6th International Conference on Coordination Models and Languages, COORDINATION 2004. Volume 2949 of Lecture Notes in Computer Science. Springer-Verlag, Pisa, Italy (2004)
20. Carbunar, B., Valente, M.T., Vitek, J.: Coordination and mobility in CoreLime. Math. Struct. in Comp. Science **14** (2004) 397–419
21. Bruni, R., Montanari, U.: Concurrent models for Linda with transactions. Math. Struct. in Comp. Science **14** (2004) 421–468
22. Charles, A., Menezes, R., Tolksdorf, R.: On the implementation of SwarmLinda. In: ACM-SE 42: Proc. 42nd Annual Southeast Regional Conference, New York, NY, USA, ACM Press (2004) 297–298
23. Lucchi, R., Zavattaro, G.: WSSecSpaces: a secure data-driven coordination service for web services applications. In: SAC '04: Proc. 2004 ACM Symposium on Applied Computing, New York, NY, USA, ACM Press (2004) 487–491
24. Cheung, L., Kwok, Y.: On load balancing approaches for distributed object computing systems. J. Supercomput. **27** (2004) 149–175
25. Wells, G.: An implementation of Linda. In Cilliers, C., ed.: Proc. Fifth Computer Science Research Students' Conference, Katberg (1990) 302–307
26. Clayton, P., de Heer Menlah, F., Wells, G., Wentworth, E.: An implementation of Linda tuple space under the Helios operating system. South African Computer Journal **6** (1992) 3–10
27. Rehmet, G.: Remora: Implementing adaptive parallelism on a heterogeneous cluster of networked workstations. Master's thesis, Rhodes University (1995)
28. Clayton, P., Rehmet, G.: Implementing adaptive parallelism on a heterogeneous cluster of networked workstations. In: Proc. 1995 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'1995). (1995) 571–580
29. Wells, G., Chalmers, A.: Extensions to Linda for graphical applications. In: Proc. International Workshop on High Performance Computing for Computer Graphics and Visualisation. (1995) 174–181 Reprinted in [47].
30. Wells, G., Chalmers, A., Clayton, P.: An extended version of Linda for Transputer systems. In O'Neill, B., ed.: Parallel Processing Developments (Proc. 19th World Occam and Transputer User Group Technical Meeting), IOS Press (1996) 233–240
31. Wells, G., Chalmers, A.: An extended Linda system using PVM. In: Proc. 1995 PVM Users' Group Meeting. (1995) URL: `http://www.cs.cmu.edu/Web/Groups/-pvmug95.html`.
32. Wells, G., Chalmers, A., Clayton, P.: An extended version of Linda for distributed multimedia applications. In: Proc. SAICSIT '99. (1999)
33. Wells, G., Chalmers, A., Clayton, P.: A comparison of Linda implementations in Java. In Welch, P., Bakkers, A., eds.: Communicating Process Architectures 2000. Volume 58 of Concurrent Systems Engineering Series. IOS Press (2000) 63–75
34. Wells, G., Chalmers, A., Clayton, P.: Extending Linda to simplify application development. In Arabnia, H., ed.: Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001). CSREA Press (2001) 108–114
35. Wells, G., Chalmers, A., Clayton, P.: Extending the matching facilities of Linda. In Arbab, F., Talcott, C., eds.: Proc. 5th International Conference on Coordination Models and Languages (COORDINATION 2002). Volume 2315 of Lecture Notes in Computer Science., Springer (2002) 380–388
36. Wells, G., Chalmers, A., Clayton, P.: Linda implementations in Java for concurrent systems. Concurrency and Computation: Practice and Experience **16** (2004) 1005–1022

37. Wells, G.: New and improved: Linda in Java. In Gibson, P., Power, J., Waldron, J., eds.: Proc. Third International Conference on the Principles and Practice of Programming in Java (PPPJ 2004). ACM International Conference Proceedings Series, Las Vegas (2004) 67–74.
38. Wells, G.: A Programmable Matching Engine for Application Development in Linda. PhD thesis, University of Bristol, U.K. (2001)
39. Akhurst, T.: The role of parallel computing in bioinformatics. Master's thesis, Rhodes University (2004)
40. Wells, G., Akhurst, T.: Using Java and Linda for parallel processing in bioinformatics for simplicity, power and portability. In: Proc. IPS-USA-2005, Cambridge, MA, USA (2005)
41. Sun Microsystems: Java Media Framework API. (URL: `http://java.sun.com/-products/java-media/jmf/index.html`)
42. Wells, G.: New and improved: Linda in Java. Science of Computer Programming (2005) In press.
43. Asaravala, A.: Giving SOAP a REST. URL: `http://www.devx.com/DevX/-Article/8155` (2002)
44. McMillan, R.: A RESTful approach to web services. URL: `http://-www.networkworld.com/ee/2003/eerest.html` (2003)
45. IBM: The TSpaces programmer's guide. (URL: `http://www.almaden.ibm.com/-cs/TSpaces/html/ProgrGuide.html`)
46. Foster, M., Matloff, N., Pandey, R., Standring, D., Sweeney, R.: I-Tuples: A programmer-controllable performance enhancement for the Linda environment. In Arabnia, H., ed.: Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001). CSREA Press (2001) 357–361
47. Chen, M., Townsend, P., Vince, J., eds.: High Performance Computing for Computer Graphics and Visualisation. Springer-Verlag (1996)