Extending the Matching Facilities of Linda

George Wells[1], Alan Chalmers[2], and Peter Clayton[1]

[1] Department of Computer Science, Rhodes University,
Grahamstown, 6140, South Africa
{G.Wells, P.Clayton}@ru.ac.za
[2] Department of Computer Science, University of Bristol,
Bristol BS8 1UB, U.K.
alan@compsci.bristol.ac.uk

Abstract. This paper discusses the associative matching mechanism used in the Linda coordination language for the retrieval of data. There are a number of problems with this mechanism which are discussed in the light of the requirements of applications using Linda. A number of solutions to these problems have been proposed. These are discussed and compared with a new approach to solving these problems. The benefits and the limitations of the new approach are considered, showing how it provides a considerable improvement in this area.

1 Introduction

The Linda[1] coordination language for distributed and parallel programming was first proposed by David Gelernter in the mid-1980's[1]. In recent years it has become a popular approach for handling the coordination of distributed processes in Java[2] programs, with a number of commercial products and research projects based on the Linda model[2–16]. Many of these systems have attempted to address a number of problems arising from the simple associative matching mechanism that is used by Linda for the retrieval of data.

The first section of this paper presents a brief discussion of the problems related to the use of the associative matching mechanisms in Linda (full details of the Linda programming model may be found in [17]). This is followed by a survey of Java implementations of Linda which have embodied various solutions to the matching problems. It then describes our work on an extended version of the Linda model called *eLinda*. The eLinda system contains a number of extensions, including the *Programmable Matching Engine*, intended to help overcome the deficiencies of the conventional Linda associative matching technique.

2 Problems with the Associative Matching Mechanism

The associative matching mechanism used for the retrieval of tuples in Linda works very well in many situations, such as simple one-to-one communication,

[1] Linda is a registered trademark of Scientific Computing Associates.
[2] Java is a registered trademark of Sun Microsystems, Inc.

one-to-many communication, implementing semaphores, barrier synchronisation, etc. However there are situations where this simple matching is not adequate.

As a simple example, consider a set of tuples, where an application needs to locate the tuple with the maximum 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, search through them for the one with the maximum value, and then return the tuples to the tuple space. During this procedure the tuples are not available for other processes to make use of, potentially restricting the degree of parallelism possible. Furthermore, in a distributed implementation, there is a large amount of unnecessary network traffic generated by this application.

While this is a simple example, it illustrates a general problem, namely that applications may need a "global view" of the tuples in tuple space. Other examples include finding tuples with 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 associative matching technique.

3 Extended Linda Systems

In an attempt to address the problem described in the preceding section (and others) a number of different extensions to the basic Linda approach have been proposed. This section outlines some of these projects.

3.1 TSpaces

TSpaces is a Linda-like system developed by IBM's alphaWorks research division[2, 3]. It is considerably extended from the original Linda model, particularly in terms of support for commercial applications. More importantly from our perspective, it includes several new operations, and provides a mechanism that allows applications to add new operations to the tuple space server dynamically. Each of these aspects is relevant to our discussion and will be covered in more detail in the following sections.

New Operations TSpaces provides new operations for the input and output of multiple tuples, and a number of 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 registration" mechanism. This allows a process to request notification when a specified tuple is written to the tuple space or deleted from it.

Matching tuples in TSpaces can be done using so-called "indexed tuples". In this case the fields of tuples may be named, ranges of values may be included in the matching process, and AND and OR operations may be specified. These

features may all be used in combination. It is also possible to perform matching on XML[3] data contained in tuples.

Extending the Server New commands can be added to the TSpaces system relatively easily. The implementation of TSpaces makes use of a number of layers of software. At the lowest level the tuples themselves are stored in a form of database. This may be an actual DBMS product, or a data structure in memory. Above this is the tuple management layer, which handles the retrieval of tuples from the database. Above this layer, accessed through a well-defined programming interface, is the operator management level. This is comprised of a number of "factory" objects arranged in a list. The factories are responsible for creating "tuple handlers" for each command that is passed to the tuple space. If a factory does not recognise a particular command then it is passed down to the next factory in the list.

Users with administrator permission levels can add new factories and handlers to the system dynamically, providing a great deal of flexibility. However, this is a complex process from a programmer's perspective, as has also been noted by Foster *et al*[13].

Application of the Extensions The extended features of TSpaces provide for a somewhat better solution to the "maximum" matching problem outlined in Section 2. In this case either the `scan` or the `consumingScan` operation provided by TSpaces can be used to retrieve all of the tuples in one step. The application can locate the required tuple, and then return the tuples to tuple space if necessary. While the end effect is exactly the same as in the original Linda model, the application is simplified through the use of these new operations.

The extended matching facilities in TSpaces (allowing ranges of values, XML-matching, etc.) may be useful in that they cover a number of common situations. However, they do not provide a general solution for the matching requirements of all applications. Notably, they do not address the "maximum" problem described in Section 2.

The possibility of extending the TSpaces server to include specialised matching commands also exists. For our example problem, this would entail writing a new command handler to search through the tuple space and return the one with the maximum value. However, there are a number of drawbacks to this approach:

- The users of applications requiring new matching operations will need to be granted administrator rights (or the system administrator must install all new matchers).
- From a system design perspective, modifying the server in order to support specific applications may be undesirable. Ensuring that different new com-

[3] Extensible Markup Language, a specification for structured documents produced by the World Wide Web Consortium[18].

mands do not interfere with each other in unintended ways may also be problematic.
- The new commands are awkward to use.
- As already noted, the process of adding new command handlers is not simple.

3.2 XMLSpaces

With the wide adoption of XML in the computer industry, it is increasingly useful for XML support to be provided by a Linda system. XMLSpaces was designed as an extension of TSpaces to address the limited facilities that it has for matching XML-formatted data[4]. As such, XMLSpaces does not attempt to provide a general solution to the matching problem, but one that is aimed at a specific application area.

The XML support in XMLSpaces is provided through the object-oriented features of the Java programming language. The `Field` class used by TSpaces for the fields in tuples is subclassed to create a new class called `XMLDocField`. This class overrides the matching method used by TSpaces to provide matching on the basis of the XML content of the field. The matching is performed by a method of the anti-tuple that can be provided by the application programmer. This results in a great deal of flexibility for XML matching operations. A number of matching operations are currently supported, including the use of XML query languages, such as XPath[19].

3.3 The Work of the York Coordination Group

The coordination research group at the University of York has been actively researching in the area of Linda systems for some time. One of their major projects has been to extend the Linda operations with `collect` and `copy-collect`[5, 6]. These *bulk operations* may be used to move or copy multiple tuples from one tuple space to another. This provides similar functionality to the `scan` and related operations of TSpaces. While these do not directly affect the matching mechanisms, they may be used to simplify applications that need a "global view" of tuple space, such as the maximum example discussed above.

3.4 Liam

Liam is a Linda system, based on the *Chemical Abstract Machine*, or *CHAM*[7]. This is an unusual programming model, in which systems are expressed as "solutions of molecules" (multisets, describing the state of the system), and subjected to "chemical reactions" (rewriting of the multisets, subject to "reaction rules"). Programs in this model consist of sets of reaction pairs, composed of a condition, specifying when the rule may be applied, and an action, which is a function that produces new molecules from the reactants. This model is ideally suited to parallel implementation, as independent reactions may take place simultaneously.

Liam allows the matching algorithm for tuples to be provided in CHAM form. This has the drawback that programmers must become familiar with the syntax

used by the Chemical Abstract Machine. This is not a simple notation for the average application programmer to learn and use.

3.5 Objective Linda

Objective Linda is a model for object-oriented implementations of Linda[8, 9]. All aspects of an application (i.e. data, active agents and the tuple spaces) are modelled as objects. Tuple spaces form a strongly encapsulated hierarchy of objects, containing passive objects (i.e. tuples), active objects (i.e. agents) and other tuple spaces.

Of particular iterest are the extensions to the usual Linda matching mechanism. The objects that are to be used as tuples in Objective Linda are required to provide a `match` method. This method is then called when performing an input operation. This means that the programmer writing the classes to be used as tuples in an application can define the precise meaning of a "match".

This feature of Objective Linda provides a restricted form of extended matching similar to that in the eLinda system. The strengths and weaknesses of this approach will be discussed at the end of the next section.

3.6 CO^{3}PS

CO^{3}PS stands for "Computation, Coordination and Composition with Petri net Specifications"[10, 11]. The coordination model used in CO^{3}PS is based closely on that of Objective Linda. The main application of extended matching in CO^{3}PS is to support the introduction of *non-functional requirements* into the design of a system. Examples of such requirements are efficiency, load-balancing and security.

In order to support this design technique, CO^{3}PS makes use of a *reflective architecture*. This is an architecture that permits the designer to reflect on the behaviour of the system, and to adapt it, without affecting the interaction with clients. The developers of CO^{3}PS emphasize that this should be done without impacting on the semantics of the coordination operations.

Discussion of Objective Linda and CO^{3}PS Both Objective Linda and CO^{3}PS allow the matching method for tuples to be overridden. This is rather counter-intuitive, in that the matching is effectively provided by the *tuple*, rather than the anti-tuple. This has the implication that programmers writing classes for tuples need to consider how they may be retrieved, while it is the anti-tuples that are used for input operations in Linda. Furthermore, this makes it extremely difficult to apply different matching criteria to a single type of tuple at different times (or in different applications). Associating the matching logic with the anti-tuples is thus a more natural approach.

Matching is also constrained to a one-to-one situation: the `match` method is called to determine whether the tuple matches a single anti-tuple. Thus there is no way of providing operations that aggregate tuples to form a result.

In CO^{3}PS this approach has been adopted for the reflective architecture, providing for the non-functional requirements of an application. The key to this philosophy is that the semantics of the coordination operations may not be altered by the imposition of non-functional requirements. However, in many situations it is very useful to be able to relax or alter the semantics of the matching operations (such as aggregating multiple tuples to form a result).

3.7 ELLIS

ELLIS (EuLisp LInda System) is a Linda system developed in EuLisp[12]. Of particular interest is that matching is performed by a method in the tuple space class. This allows the matching method to be overridden, but the mechanism seems clumsy: new classes of tuple spaces must be created to support new matching algorithms. Few details of this process are given in the description of ELLIS, but the programming interfaces for new matchers appear to be complex and to involve dealing with the pool of tuples at a very low level of abstraction.

3.8 Summary

What all of these projects indicate is an underlying weakness of the basic Linda associative matching technique. While each of these systems has addressed this problem in one way or another, they each have their own deficiencies. The next section presents the eLinda system and the extensions to the associative matching procedure, which overcomes these problems.

4 eLinda

The eLinda system is based closely on the standard Linda model. It uses a fully-distributed tuple space model where any tuple may reside on any node. This poses particular problems for matching, in that many processing nodes may be required to participate in a matching operation. Testing has shown that the performance of eLinda is on a par with that of other Java Linda systems (such as TSpaces and JavaSpaces[20]), but that Java is currently not a viable platform for parallel processing applications[21].

The eLinda system contains three extensions to the Linda programming model: a "broadcast" output operation, multimedia support and the *Programmable Matching Engine*. The focus of this paper is on the last of these. Further details of the other features can be found in [22].

4.1 The Programmable Matching Engine

The Programmable Matching Engine (or *PME*) allows the use of more flexible criteria for the associative addressing of tuples. This is useful in situations such as that exemplified by finding the tuple with the maxiumum 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 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 maximum tuple, each section of the tuple space would be searched locally for the largest tuple and that returned to the matcher running in the originating process, which would then select the largest of all the replies received. This process is completely transparent to the application, which simply inputs a tuple, using a specialised matcher. From the application programmer's perspective this could be expressed simply as `in.maximum(?field1, ?=field2)`. The notation that is used is to follow the Linda input operation with the name of the matcher to be used. 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, or the York bulk operations.

New matchers are written as Java classes. They can make use of a simple library that provides controlled access to tuple space, communication mechanisms, etc.

4.2 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 need to be gathered together 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. However, it would be incorrect to believe that the PME is limited to these—it is just as applicable to textual, XML or other problem domains.

5 Conclusions

The significant number of projects that have, in one way or another, extended the functionality of the matching operations in Linda points to the weakness that

is embodied in the original programming model. The nature of the extensions ranges from those exemplified by TSpaces, where the tuple space server itself is reconfigured to support new operations, to systems like XMLSpaces, Objective Linda and Liam where the matching process is specified by overriding the matching method used, in some cases for very specific purposes.

What is found in comparing these proposals with the eLinda PME is that the PME proposal can emulate all of these alternative approaches. Furthermore, the PME approach allows for a range of tuple space implementation techniques, ranging from fully distributed to centralised, whereas most of the other systems are implemented only for centralised configurations.

In many cases the PME approach is more intuitive and elegant than the alternatives. The approach adopted in Objective Linda and CO^{3}PS, where the matching method in an object/tuple can be overridden, fits well with the object-oriented philosophy of Java. However, it limits matching operations to one-to-one situations, and fixes the matching possibilities at the time that the tuple class is written. By providing the matcher as an independent object, the PME approach opens up the possibilities of aggregating operations, and provides the flexibility of being able to apply many matchers to a single type of tuple. The approach used for adding new commands in TSpaces, while providing a high degree of flexibility, effectively requires the reconfiguration of the tuple space server to support new operations. This is complex and potentially dangerous in a multiuser situation.

5.1 Future Work

A number of Linda systems have made extensions in the area of *output* and *update* operations, similar in some respects to those described above for input operations. [2, 13, 14]. This again points to a weakness in the Linda programming model, and it appears that an approach analagous to that of the Programmable Matching Engine may be beneficial in such situations too.

Acknowledgments

This work was supported by the Distributed Multimedia Centre of Excellence in the Department of Computer Science at Rhodes University, with funding from Telkom SA, Lucent Technologies, Dimension Data and THRIP.

References

1. Gelernter, D.: Generative communication in Linda. ACM Trans. Programming Languages and Systems **7** (1985) 80–112
2. IBM: TSpaces. (URL: `http://www.almaden.ibm.com/cs/TSpaces/index.html`)
3. Wyckoff, P., McLaughry, S.W., Lehman, T.J., Ford, D.A.: T Spaces. IBM Systems Journal **37** (1998) 454–474

4. Tolksdorf, R., Glaubitz, D.: Coordinating web-based systems with documents in XMLSpaces. URL: `http://flp.cs.tu-berlin.de/~tolk/xmlspaces/-webxmlspaces.pdf` (2001)
5. Butcher, P., Wood, A., Atkins, M.: Global synchronisation in Linda. Concurrency: Practice and Experience **6** (1994) 505–516
6. Rowstron, A., Wood, A.: Solving the Linda multiple rd problem. In Ciancarini, P., Hankin, C., eds.: Coordination Languages and Models, Proc. Coordination '96. Volume 1061 of Lecture Notes in Computer Science., Springer-Verlag (1996) 357–367
7. Campbell., D.: Constraint matching retrieval in Linda: extending retrieval functionality and distributing query processing. Technical Report YCS 285, University of York (1997)
8. Kielmann, T.: Objective Linda: A Coordination Model for Object-Oriented Parallel Programming. PhD thesis, University of Siegen, Germany (1997)
9. Kielmann, T.: Object-Oriented Distributed Programming with Objective Linda. In: First International Workshop on High Speed Networks and Open Distributed Platforms, St. Petersburg, Russia (1995)
10. Holvoet, T., Berbers, Y.: Reflective programmable coordination media. [23] 1236–1242
11. Holvoet, T.: An Approach for Open Concurrent Software Development. PhD thesis, Department of Computer Science, K.U.Leuven (1997)
12. Broadbery, P., Playford, K.: Using object-oriented mechanisms to describe Linda. In Wilson, G., ed.: Linda-Like Systems and Their Implementation. Technical Report 91-13. Edinburgh Parallel Computing Centre (1991) 14–26
13. Foster, M., Matloff, N., Pandey, R., Standring, D., Sweeney, R.: I-Tuples: A programmer-controllable performance enhancement for the Linda environment. [23] 357–361
14. Rowstron, A.: Mobile co-ordination: Providing fault tolerance in tuple space based co-ordination languages. URL: `http://www.research.microsoft.com/~antr/-papers/mobile.ps.gz` (1999)
15. Sudell, A.: Design and implementation of a tuple-space server for Java. URL: `http://www.op.net/~asudell/is/linda/linda.html` (1998)
16. Smith, A.: Towards wide-area network Piranha: Implementing Java-Linda. (URL: `http://www.cs.yale.edu/homes/asmith/cs690/cs690.html`)
17. Carriero, N., Gelernter, D.: How to Write Parallel Programs: A First Course. The MIT Press (1990)
18. World Wide Web Consortium: Extensible markup language (XML). (URL: `http://www.w3.org/XML`)
19. World Wide Web Consortium: XML Path language (XPath) version 1.0. W3C Recommendation, URL: `http://www.w3.org/TR/xpath.html` (1999)
20. Freeman, E., Hupfer, S., Arnold, K.: JavaSpaces Principles, Patterns, and Practice. Addison-Wesley (1999)
21. 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
22. Wells, G.: A Programmable Matching Engine for Application Development in Linda. PhD thesis, University of Bristol, U.K. (2001)
23. Arabnia, H., ed.: Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001). CSREA Press (2001)