A Tuple Space Web Service for Distributed Programming

George C. Wells
Department of Computer Science, Rhodes University
Grahamstown, 6140, South Africa

Abstract

This paper describes the implementation of a web service providing a tuple space service for distributed programming applications. Previous research has established the benefits of using tuple space-based systems, particularly with regard to simplicity. This project has developed a new web service providing a tuple space mechanism for distributed applications based on web services. The approach that has been adopted makes use of REST (Representational State Transfer) for the web service. Initial results of testing the system for a bioinformatics application indicate that the project has provided an efficient, useful, easy-to-use mechanism for distributed web-service applications.

Keywords: Distributed processing, Web services, REST, Tuple space, Linda

1 Introduction

The use of a shared tuple space to simplify interprocess communication in parallel/distributed systems was introduced by David Gelernter and his colleagues at Yale University when they developed the Linda[1] coordination language in the mid-1980's[1]. After an initial surge of interest in this model, interest waned during the early 1990s. However, recently interest has been renewed in this approach, particularly in the Java community where a number of projects, both commercial and research, have explored the use of various tuple space models. Notably, Sun Microsystems developed the JavaSpaces specification[2]. IBM have also produced a commercial Linda implementation in Java, called TSpaces[3]. Further details of the Linda/tuple space programming model may be found in [4].

Several academic research projects have also studied various aspects of the Linda/tuple space approach to parallel/distributed processing (see the partial list in [5]). Our own prior research has concentrated on improving the efficiency of the associative matching mechanisms in Linda[6]. Arising from this research, we have considered the possibility of providing a Linda/tuple space web service[7]. This topic has also been studied by a few other groups, notably Lucchi and Zavattaro[8], who focus specifically on the security of a tuple space web service, and the Advanced Information Systems Group at the Universidad de Zaragoza[9].

This paper describes the implementation of a Linda web service, using the Representational State Transfer (REST) approach, proposed by Roy Fielding[10]. This is distinguished by its simplicity, in comparison to more conventional approaches such as the use of SOAP, WSDL, etc.[11].

The remainder of this paper consists of a brief introduction to the REST approach to the development of web services. This is followed by a description of the implementation of our tuple space web service, and the results of our initial evaluation of this project.

2 REST

Web services have been proposed as a mechanism for utilising the ubiquity of the World Wide Web to provide a simple, convenient distributed service architecture. Some of this promise has been dispelled by the proliferation of numerous, complex standards for various aspects of web service provision. For example, Sean Rhody, editor of the Web Services Journal, identified the problem as follows "There are too many

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

Table 1: Examples of the XML Format of eLindaWS Tuples

```
<?xml version="1.0" encoding="UTF-8"?>
<OutputTuple ...>
  <TupleData>
    <Field type="string">point</Field>
    <Field type="int">12</Field>
    <Field type="int">67</Field>
  </TupleData>
</OutputTuple>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<InputTuple blocking="true" destructive="true" ...>
  <TupleData>
    <Field type="string">point</Field>
    <Field type="int"></Field>
    <Field type="int"></Field>
  </TupleData>
</InputTuple>
```

Note: schema specifications have been omitted for brevity.

standards, at too granular a level, and too much competition"[12]. Foremost among the standards organisations responsible for this proliferation is the Web Services Interoperability Organization (WS-I). Also contributing to the plethora of standards are OASIS, the W3C and Microsoft. In 2004, Tim Bray, one of the originators of the XML specification, produced a table summarising the standards produced by these organisations[13]. He broke the standards down into eight groups, with a total of 33 standards, comprising 783 pages of specification.

Partly as a reaction to this proliferation of standards, Roy Fielding proposed *Representational State Transfer* (REST) as a simpler alternative for the implementation of web services. The REST approach essentially calls for the transmission of simple XML messages across the Internet, using the standard HTTP protocol[10]. The simplicity of this approach is highly appealing, and it was selected for the implementation of our tuple space web service, which is described in the next section.

3 The eLinda Web Service

Our original research into tuple space-based distributed programming mechanisms was named eLinda (for extended-Linda) and we have kept this name for the new web service implementation. Our web service is also referred to as *eLindaWS* for brevity. It consists of two separate parts: a server-side component, providing the web service itself, and a small client library, abstracting some of the details of the use of the web service.

The XML schema used for eLindaWS is fairly simple, defining two main forms of message: an `InputTuple` and an `OutputTuple`. Examples are shown in Table 1. An `OutputTuple` corresponds to the form of tuple produced by the `out` operation, with all fields having defined values. An `InputTuple` corresponds to an *antituple*, and may contain unspecified fields (indicated by empty XML elements) for associative matching. Attributes of the `InputTuple` element specify the exact form of the input operation (i.e. whether predicate or blocking, and whether the tuple is removed from the tuple space by the operation or not).

3.1 The Web Service

The server side of the web service has been developed as a Java servlet[14], using the Apache Tomcat servlet container[15]. Servlets are a Java technology for providing dynamic web content, and are thus ideally suited for the implementation of web services. The servlet receives an XML "message" and parses this using the Streaming API for XML (StAX)[16]. StAX is a "pull parser" that provides simple, powerful and efficient facilities for parsing and producing XML under the control of the program (rather than the parser). The resulting Java data structure is then used with a Linda-style tuple space to carry out the relevant input or output operation. Blocking input operations result in the thread that is executing the servlet blocking, and thus no response is generated until the underlying tuple space input operation is satisfied.

Given the submission-response mechanism used for dynamic web content, all web service operations result in a response being sent back to the client of the web service. In the case of output operations, or negative responses to predicate input operations, an empty `OutputTuple` (i.e. with no `TupleData` or `Field` elements) is returned. In the case of blocking input operations or successful predicate input operations a complete `OutputTuple` is returned.

An additional benefit of the use of a servlet container for eLindaWS is that debugging becomes very simple. In particular, a second servlet was developed that obtains a listing of the contents of the tuple space and returns this as a conventional (HTML) web page. Viewing this page in a web browser produces a very useful snapshot of the current state of the tuple space.

Table 2: Client Library Interface

```
public class TSClient
  { public TSClient (String url);
    public TSClient ();
    public void out (Object... tuple);
    public Object[] in (Object... antituple);
    public Object[] inp (Object... antituple);
    public Object[] rd (Object... antituple);
    public Object[] rdp (Object... antituple);
  } // class TSClient
```

3.2 The Client Library

In order to simplify the development of clients making use of the eLinda web service, a client-side "wrapper" has been produced that provides the same interface as a typical Linda tuple space implementation (see the class outline in Table 2). This provides facilities to bind to a web service (either explicitly named or using a default service), and support the basic Linda input/output operations. Note the use of the new "varargs" feature of Java 5.0, to simplify the variable-length parameter lists, and leading to a very natural syntax for the input/output operations.

Tuples are represented by simple arrays of objects. Use of the methods provided by this interface results in an appropriate XML document (`InputTuple` or `OutputTuple`) being generated (using the StAX XML generation mechanisms), and then sent to the web service (using a simple HTTP client library, provided by the Apache Software Foundation as part of the Jakarta Commons project[17]). A response is then awaited — either a simple acknowledgment, or else a result tuple for successful input operations.

For this version of our system we have adopted the convention used by IBM's TSpaces that "wildcard" fields in tuples are specified using the `Class` object corresponding to the field's type[18]. For example, the code fragment shown below will first deposit a tuple in the tuple space and then retrieve a similar tuple (possibly the one just deposited). Note that these operations will generate the XML documents shown in Table 1.

```
TSClient ts = new TSClient("http://eval.ict.ru.ac.za/LindaWS/eLindaWS");
ts.out("point", 12, 67);
Object[] res = ts.in("point", Integer.class, Integer.class); // in("point", ?x, ?y)
```

For the current implementation, this client library provides almost total transparency of the web service (the only visible aspect is the specification of the URL for the service — see the first line of the code fragment above). However, other implementations (perhaps in other languages, or else using technologies such as Java applets) might choose, or be forced, to reveal more of the underlying implementation details.

4 Evaluation

This section presents the results of our preliminary evaluation of the new tuple space web service. The first evaluation step was a simple communication benchmark. This was followed by the implementation and performance testing of a distributed bioinformatics application (DNA sequence matching), which previously produced very good results under TSpaces[19].

4.1 Communication Benchmark Results

A simple communication benchmark has been developed, which repeatedly sends small messages[2] from one process to another and back. This allows measurement of the communication overheads introduced by the web service. The benchmark was run in a number of configurations. The first involved running the two client programs on the same computer as the web service (Machine A). This minimises the network overhead, but introduces additional process-switching overheads. The second configuration made use of two computers: a server providing the web service (Machine A), and a single client computer running both application

[2]Each message is a single tuple with one, four-character string and one integer.

Table 3: System Specifications for Testing

Machine	Specification
A, C	Intel Pentium 4, 3GHz 1GB RAM Windows XP Professional SP2
B	Intel Pentium 4, 3.2GHz 480MB RAM Windows XP Professional SP2
Network:	100MB switched UTP Ethernet network.
Java:	J2SE version 1.5.0_06

processes (Machine B). The third test was run using two separate clients (Machines C) and the same server (Machine A). The main difference in this third configuration was that there were several network switches between the clients and the server (the first tests were conducted between machines connected to the same switch). The specifications of these machines are given in Table 3.

4.1.1 Results

The results of the communication benchmarks are shown in Table 4. These results are the average of ten separate iterations of a program that exchanges 200 messages (100 in each direction).

Table 4: Results of Communication Benchmarks

Test	Time for 200 Messages	Time Per Message
1 (A–A)	1.1594s	5.80ms
2 (A–B)	1.3062s	6.53ms
3 (A–CC)	3.141s	15.7ms

It is clear from these results that the message communication times are dominated by networking issues: the best time comes from a situation where only one processor is doing all the work, but the networking is minimised, while the worst time comes in a scenario where individual processors are allocated to each process, but the network is in its least favourable configuration (multiple switch-hops). This is not too surprising, perhaps, given the communication-intensive nature of this particular benchmark. It does suggest that eLindaWS will be best suited to applications where the communication:computation ratio is low.

4.2 Bioinformatics Application

This application utilises the classic replicated-worker (or master-worker) pattern[2] to search DNA sequences for subsequences corresponding to specific proteins (or *motifs*). The motifs are expressed as regular expressions, which are then used with the Java `java.util.regex` package in order to locate the motifs within DNA sequences. More details of the bioinformatics field and the biology underlying this application may be found in [19].

4.2.1 The Structure of the Bioinformatics Application

The master process deposits tuples into the tuple space that specify the work to be done by the worker processes. Each tuple contains the name of the file containing the DNA sequence to be scanned, a unique task number and the total number of tasks generated (which may differ from the number of worker processes). The master process then retrieves the result tuples, summarises the results and reports the results and the processing time.

A worker process retrieves a task tuple and uses the task number and total number of tasks to determine which segment of the DNA sequence it is to search. The calculation of the start- and end-points of the segment also has to include a degree of overlap between adjoining sections to allow for motifs that may span

two segments. The worker process reads in the required segment of the DNA sequence from the specified file[3], which is stored locally. The regular expressions for the motifs are then used to perform the searching of the DNA segment. The workers then return the results found to the master process as tuples containing the start- and end-points of the identified motifs, and the actual DNA sequences for the motifs.

4.2.2 Results

The performance results for the bioinformatics application are shown in Figure 1. These were all obtained using Machine A as the server and varying numbers of identical client machines for the worker processes (Machine C specification), or (A–C^n) using the notation of Table 4. The DNA sequence used consisted of the first five human chromosomes, and was approximately 1GB in size. For the tests up to 15 machines, the DNA sequence was split into 15 segments (the clients were unable to handle larger segments). For the remaining tests the number of segments was set equal to the number of worker processes. Each test was run ten times, and the average taken.

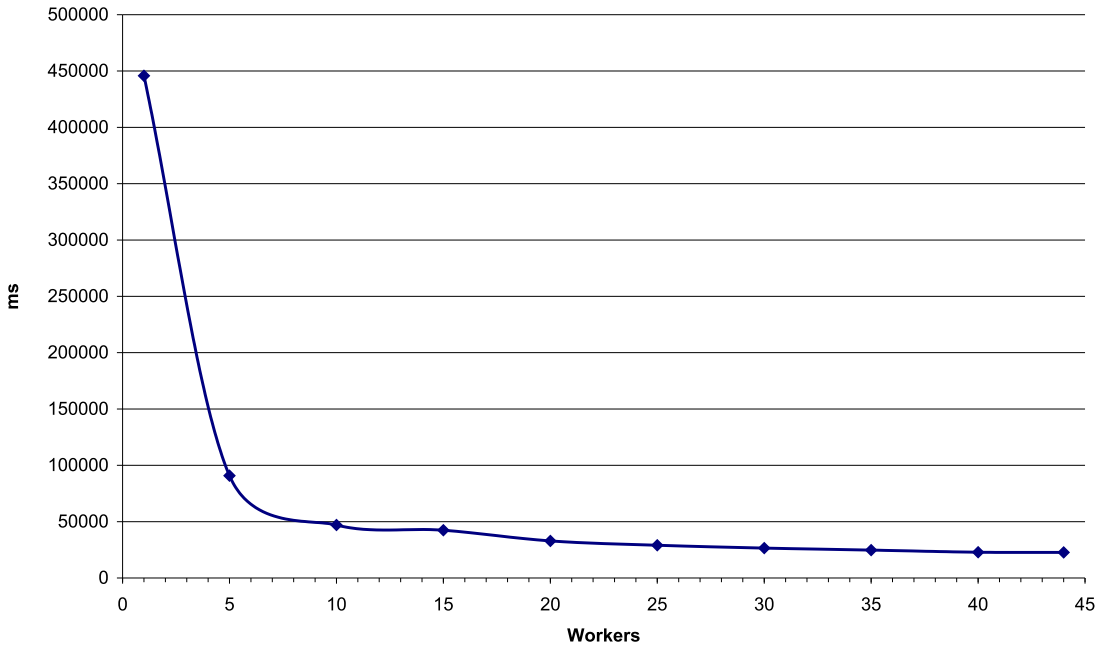

Figure 1: Execution Times for the Bioinformatics Application (1GB DNA Sequence)

Studying these results, it is clear that a small anomaly occurred for 15 workers. This is noticeable in Figure 1, and the standard deviation of the mean is an order of magnitude larger in this case. Examining the results shows that the first three results are significantly different from the remaining seven. If these three results are discarded, the overall shape of the curve in Figure 1 is smoothed and the standard deviation is reduced to the same order of magnitude as in the other cases. The reasons for this brief anomaly are not apparent, but could have been caused by some other network activity or temporary increase in the load on the server.

This minor anomaly apart, the application displays an impressive initial speedup. Using the maximum available 44 computers the application still shows speedup from the previous case (using 40 computers), but the rate of increase in the speedup is negligible. Using 20 computers the time taken is 32.8s, down from 445.8s (i.e. 7 minutes, 25.8s) — a speedup of 13.6 times, or an efficiency of 68%.

It should be noted that the testing of the bioinformatics application here used a smaller set of motifs than that reported in [19], which means the results are not directly comparable with those obtained using TSpaces.

[3]This is done using the Java NIO facilities for maximum efficiency.

5 Conclusions

One of our primary conclusions is that the development of web services using the REST approach is extremely simple. While we had some previous experience of developing dynamic web sites using servlets, which was helpful, the simplicity of the implementation phase of the prototype eLinda web service was very remarkable. Only a few days of programming were required to create the web service (and an underlying, minimal tuple space implementation) from scratch. Notably, the use of StAX made both the parsing and generation of XML documents extremely simple. It would be interesting to attempt to redevelop eLindaWS using the more conventional approaches to web services (i.e. SOAP, WSDL, etc.) for comparison.

With regard to the tuple space web service itself, our preliminary investigation has demonstrated that this provides a very simple, useful mechanism for communication and coordination in distributed systems on the Internet. For example, the bioinformatics application was rewritten from scratch to use eLindaWS in only a few days. Our previous conclusions about the benefits of tuple space approaches for distributed programming[20, 19] appear to apply just as strongly to the field of web services. The quantitative results show a good speedup for a typical bioinformatics application.

5.1 Future Work

Our prior research in the field of Linda and tuple spaces demonstrated the usefulness of permitting flexible matching criteria through the provision of a programmable matching interface[6, 20]. Integrating this mechanism with eLindaWS presents a number of challenges. First and foremost is how one can specify a matching component using the web services mechanisms. This involves sending a program component (an object as implemented in the original eLinda system) using a simple XML/HTTP communication medium. While it would be possible to send Java source code and have this compiled on the server for use in the web service, this presents obvious problems from a security perspective, and introduces potential performance overheads. Additionally, interoperability is one of the main advantages of web services[4] and this would be compromised by the adoption of a specific language for the implementation of new matchers.

An alternative might be to create a new language for the specification of new matching algorithms. This would provide portability, and with careful design could ensure suitable security. However, it is unlikely that such an approach would be able to provide the wide-ranging flexibility of a general-purpose programming language such as Java.

Some further development would also help improve the flexibility and efficiency of eLindaWS. For example, most current Linda-style systems provide for multiple, named tuple spaces allowing unrelated tuples to be stored separately, and thus providing for greater efficiency. This would be a simple extension to the current eLindaWS implementation.

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, Business Connexion, Comverse, Verso Technologies and THRIP. Financial support was also received from the National Research Foundation (NRF) of South Africa, and from Rhodes University. The author also wishes to acknowledge the wise advice and strong support of Alan Chalmers (University of Bristol) and Peter Clayton (Rhodes University) over many years. The assistance of Madeleine Wright with various aspects of web services in Java, and particularly the REST approach, has also been invaluable in developing eLindaWS.

References

[1] David Gelernter. Generative communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, January 1985.

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

[4] While the research described in this paper has made use of Java, on both the server- and client-side, there is, in principle, no reason why the components could not be implemented in different languages, as long as the XML documents exchanged adhere to the schema specification.

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

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

[5] G.C. Wells. New and improved: Linda in Java. *Science of Computer Programming*, 59(1–2):82–96, January 2006.

[6] G.C. Wells, A.G. Chalmers, and P.G. Clayton. Linda implementations in Java for concurrent systems. *Concurrency and Computation: Practice and Experience*, 16:1005–1022, August 2004.

[7] G.C. Wells. Coordination languages: Back to the future with Linda. In S. Becker, C. Canal, J.M. Murillo, P. Poizat, and M. Tivoli, editors, *Proc. 2nd International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'05)*, pages 87–98, Glasgow, July 2005. Held in conjunction with ECOOP 2005.

[8] Roberto Lucchi and Gianluigi Zavattaro. WSSecSpaces: a secure data-driven coordination service for web services applications. In *SAC '04: Proc. 2004 ACM Symposium on Applied Computing*, pages 487–491, New York, NY, USA, 2004. ACM Press.

[9] E. Mata, P. Álvarez, J.A. Bañares, and J. Rubio. Towards an efficient rule-based coordination of web services. In *IBERAMIA 2004*, volume 3315 of *Lecture Notes in Artificial Intelligence*, pages 73–82. Springer Verlag, 2004.

[10] R.T. Fielding and R.N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technology*, 2(2):115–150, May 2002.

[11] M.K. Wright. A detailed investigation of interoperability for web services. Master's thesis, Rhodes University, 2005.

[12] S. Rhody. Web service, part II: The search for the optional standards. *Web Services Journal*, 4(10), October 2004.

[13] T. Bray. WS-Pagecount. URL: `http://www.tbray.org/ongoing/When/200x/2004/09/21/-WS-Research`, October 2004.

[14] Sun Microsystems. Java servlet technology. URL: `http://java.sun.com/products/servlet/-index.jsp`, 2006.

[15] Apache Software Foundation. Apache Tomcat. URL: `http://tomcat.apache.org`, 2006.

[16] Java Community Process. JSR 173: Streaming API for XML. URL: `http://jcp.org/en/jsr/-detail?id=173`, March 2004.

[17] Apache Software Foundation. Jakarta commons. URL: `http://jakarta.apache.org/commons/-index.html`, 2006.

[18] IBM. The TSpaces programmer's guide. URL: `http://www.almaden.ibm.com/cs/TSpaces/html/-ProgrGuide.html`.

[19] G.C. Wells and T.J. Akhurst. Using Java and Linda for parallel processing in bioinformatics for simplicity, power and portability. In *Proc. IPS-USA-2005*, Cambridge, MA, USA, June 2005.

[20] 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.