C^2AS: A System Supporting Distributed Web Applications Composed of Collaborating Agents

A. I. T. Rowstron, S. F. Li and R. Stefanova
Computer Laboratory, University of Cambridge,
New Museums Site, Pembroke Street, Cambridge, CB2 3QG, UK
{aitr2, sfl20, rs10025}@cl.cam.ac.uk

Abstract

In this paper we describe the Cambridge Collaborative Agent System (C^2AS). This is a prototype system designed to demonstrate the functionality and basic architecture of a framework for co-ordination between different components (or agents) of distributed 'Web' applications.

Co-ordination in C^2AS is achieved through the use of tuple spaces, as used in Linda. However, the access primitives used in C^2AS are not those used by Linda, but the BONITA *primitives, which provide asynchronous access to tuple spaces as opposed to synchronous access. Because C^2AS is tuple space based it supports temporal and spatial separation of agents. The prototype system supports agents written in either Java (including applets) or C.*

1. Introduction

The aim of C^2AS is to develop a simple, powerful and flexible approach to allowing inter-agent communication over a Wide Area Network (WAN) such as the World Wide Web (WWW). In order to achieve this C^2AS does not make any assumptions about the programming language being used. An agent written in C running in Cambridge, UK should be able to co-ordinate with an agent written in Java running as an applet on a Netscape Web browser in Cambridge, MA, USA. Indeed, the two agents should have no idea of the physical distance between them, or that they are written in different languages. The current prototype supports process co-ordination between agents written in Java and C (and C++). The C agents are compiled into native machine code and do not use the Java Virtual Machine.

In order to demonstrate the attributes of C^2AS and how distributed Web applications can use it, a specific example of a Web application is considered; a Virtual University.

The proposal for the use of tuple spaces for geographically distributed systems is not a novel one, indeed Gelernter[5] generally talked about the concept of mirror worlds built using tuple spaces. More recently, the PageSpace project[3] has actually implemented such a system. The differences between the PageSpace and C^2AS are: C^2AS is designed to support agents written in different languages; the access primitives to the tuple space are different; multiple tuple spaces are supported; and the underlying architecture is radically different.

Ciancarini et al.[2] describe in detail the current problems with the Web as currently used. The problems can essentially be described as the need for temporal separation[1] and peer-to-peer communication, removing the need to create special servers, MIME-types, helper applications and so on for an application that requires spatial separation.

2. C^2AS functionality and architecture

C^2AS uses tuple spaces as used in Linda[1]. The properties of tuple spaces have for over a decade been used in parallel computing and Local Area Network (LAN) based computing. C^2AS represents an extension of the ideas of tuple spaces to allow distributed Web based applications. Within the system *the only* way two processes can communicate is via a tuple space. C^2AS *does not* support direct communication between two agents.

Earlier work has shown that the Linda primitives for accessing tuple spaces may not be appropriate for Web based applications (applications with the potential for a large geographic distribution of communicating agents). This is fundamentally because the primitives provide only *synchronous* communication *with tuple spaces*. Thus, the C^2AS uses the BONITA[13] primitives which allow both synchronous and asynchronous access to tuple spaces.

The current implementation of C^2AS provides all the co-ordination functionality envisaged. However, the underlying run-time system is based on a LAN run-time system

[1]The ability for two agents to co-ordinate despite *not* executing concurrently

and is not scalable. The current prototype system does not address the problems of security or agent migration.

The co-ordination functionality of C^{2}AS is now considered, and then the prototype architecture is described.

2.1. C^{2}AS Functionality

As with Linda the fundamental objects of C^{2}AS are tuples, templates and tuple spaces:

Tuple A tuple is an ordered collection of fields. Each field has a type and a value associated with it. A field with both a value and a type is known as an actual. The same field can be replicated many times within a tuple. The tuple: $\langle 10_{int}$, "Hello World"$_{string}$, 10_{int}, 'A'$_{char}\rangle$ is a tuple containing four fields with the type of the field shown as a subscript of the value. The types of the fields are normally restricted by the language into which Linda is embedded. Tuples are placed into tuple spaces and are removed from tuple spaces using an a associative matching process.

Template A template is similar to a tuple except the fields do not need to have values associated with them, but all fields must have a type. A field that has only a type and no value is known as a formal, and a template is a tuple which can have formals.

The templates: $\langle| 10_{int}$, "Hello World"$_{string}$, 10_{int}, 'A'$_{char}|\rangle$ and $\langle| 10_{int}$, $\Box_{string}$, $\Box_{int}$, 'A'$_{char}|\rangle$ will match the tuple $\langle 10_{int}$, "Hello World"$_{string}$, 10_{int}, 'A'$_{char}\rangle$. In this paper the symbol $\Box$ in a template is used to indicate that the field is a formal, so it has no value.

Tuple space A tuple space is a *logical shared associative memory* that is used to store tuples. A tuple space implements a *bag* or *multi-set*, and the same tuple may be present more than once and there is no ordering of the tuples in a tuple space.

Tuples are inserted into tuple spaces. In order to retrieve a tuple an associative match is performed between a template and the tuples in the tuple space. The choice of data types that can be inserted into a tuple has been restricted because the agents can be written in different languages, which support different types. Currently, C^{2}AS supports typed objects of integer, character, and string. This means that entire Java objects cannot be inserted into a tuple. There is no reason why new types cannot be added, indeed there is nothing to stop language dependent typed objects being inserted in tuples, but it means that only agents written in the same language can access these tuples.

A number of access primitives are embedded into the host languages, in the case of the C^{2}AS prototype, C and Java. C^{2}AS uses the BONITA[13] primitives:

rqid = dispatch(ts, tuple | [template, destructive | nondestructive]) This is an overloaded primitive which controls all accesses to a tuple space which require a tuple to be either placed into a tuple space or removed from a tuple space. The tuple space to be used is `ts`. If a `tuple` is specified then this tuple is placed in the tuple space. If a `template` is specified then this indicates that a tuple is to be retrieved from the specified tuple space. If this is the case then an extra field is used to indicate if the tuple retrieved should be removed (`destructive`) or not removed(`nondestructive`) from the tuple space `ts`. This primitive is non-blocking and returns a *request identifier* (**rqid**) which is subsequently used with other primitives to retrieve the matched tuple.

rqid = dispatch_bulk(ts1, ts2, template, destructive | nondestructive) This initiates the movement of tuples between tuple spaces. Tuple space `ts1` is the source tuple space and the destination tuple space is `ts2` and the tuples are either moved (`destructive`) or copied (`nondestructive`). A count of the number of tuples copied or moved is returned. Again this is achieved by the primitive returning a *request identifier* (**rqid**) which is subsequently used with other primitives to get a count of the number of tuples moved or copied. The number of tuples moved or copied depends on the stability of the tuple space. If the tuple space is stable (there are no operations which are destructive being performed in parallel with this primitive) then all the available tuples are copied or moved. The semantics of the primitive, when other primitives are being performed concurrently can be found in the description of the `copy-collect` primitive in Rowstron[13]. The primitive is non-blocking.

arrived(rqid) This primitive checks to see if the result associated with rqid is available. If the result is available then the primitive returns the result. The primitive is *non-blocking* and returns false if the result associated with rqid is not yet available. The result will either be a tuple (the result of a `dispatch`) or an integer (the result of a `dispatch_bulk`).

obtain(rqid) This primitive is similar to the `arrived` primitive except it blocks waiting for the result associated with rqid if the result is not available immediately.

ts = tsc This primitive is used to create a new tuple space within the system. The tuple space is guaranteed to be unique, and a handle to the tuple space is returned.

The exact syntax of each of the primitives depends upon the host language being used. It should be noted that the Linda primitives of `in`, `rd` and `out` can all be emulated using the BONITA primitives[13]. An `in` primitive is a blocking primitive that removes an arbitrary matching tuple from a tuple space, blocking if there are no matching tuples. A `rd` primitive is an `in` primitive that returns a copy of the tuple but does not remove the tuple from the tuple space. An `out` primitive inserts a tuple into a tuple space.

C^{2}AS uses the BONITA primitives rather than the Linda primitives because they provide asynchronous access to *tuple spaces*. Tuple spaces provide asynchronous inter-agent communication, but each agent can only access a *tuple space* in a synchronous manner. Rowstron et al.[13] describe the overhead costs of performing an `in` or `rd` primitive. The drawback, in geographically distributed systems, of synchronous tuple space access is that the primitives *block* regardless of whether the matching tuple is present or not, simply due to the time taken for a message to reach the system managing the tuples and the time taken for the reply message. The agent (or in Java case the thread) which performs the Linda primitive will block for this communication time.

2.2. Prototype architecture

The architecture of the prototype system is shown in Figure 1.

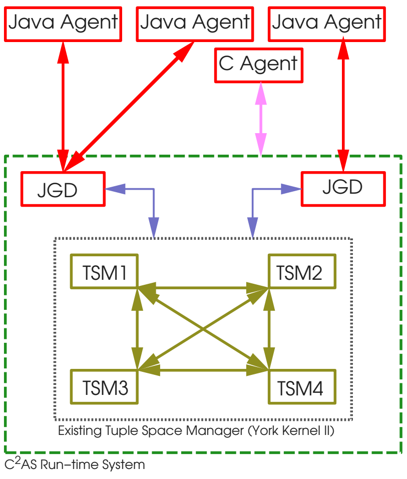


Figure 1. Architecture of prototype C^{2}AS.

The C^{2}AS architecture was initially created to be simple. The underlying storage mechanism (the tuple space manager or kernel) is based on the York Kernel II[9, 11, 12]. The kernel is distributed over a number of workstations, with each workstations running a process *TSMx*. Within the kernel every tuple space is distributed over potentially *all* TSM nodes. Therefore, each node may contain tuples that belong to the same tuple space. This requires under many circumstances arbitration, and subsequently all nodes must be able to communicate with each other. This leads to a fixed tuple space manager architecture which cannot expand (or shrink) over time. The kernel is built upon PVM[16].

Within the current prototype architecture C based agents communicate with the TSM nodes directly, using PVM and therefore, can only be executed on workstations included within the PVM virtual machine that the kernel is using.

In order to enable Java based agents to communicate with the kernel, another server process (written in Java) is started on each workstation which is running part of the kernel, called the Java Gateway Daemon (JGD). A Java agent can connect with *any* of the JGDs using a dedicated socket. Because there are many JGDs and the Java agent can connect with any of them the JGDs do not become a bottleneck within the system. When the agents are not written as applets this is fine, however due to security restrictions within the Java Virtual Machine when embedded in WWW browsers this is a problem for applets. An applet can only open a socket to the same machine which the Web server that provided the applet is executing on. Currently, in order to overcome this, a small Web server is also run on every workstation that runs one of the JGD. When the level of Java security is configurable, as planned, this restriction may be removed, or otherwise a simple Web server will be folded into the JGD, to allow it to process HTTP requests. However, even if a HTTP server is integrated with the JGDs a dedicated socket will still be used for communication between the agents and the JGD. We anticipate a performance advantage by maintaining an open socket because of the TCP/IP slow start policy.

2.3. Example

Figure 2 and Figure 3 show two small and simple agents, one written in C and the other in Java. The C agent inserts 100 tuples into a tuple space, and the Java agent retrieves the tuples and prints the second field of each tuple to the screen. Both use a tuple space which is called GTS, or the global tuple space. Traditional Linda systems support a global tuple space that all processes (agents) can access. C^{2}AS supports this global tuple space, but processes can also create new tuple spaces.

```
int main(int argc, char *argv[]) {
  int x;
  for (x = 0; x < 100; x++)
    dispatch(gts, L_INT x, L_INT (x*x), L_END);
  return 0;
}
```

Figure 2. A C agent that inserts 100 tuples into a tuple space.

Figure 3 shows the Java agent. In the Java embedding of the primitives the global tuple space handle has to be explicitly created, and this is done on line 7. The object `gts` is an instantiation of the class TupleSpace, whose methods include the BONITA primitives. Therefore, executing `gts.dispatch` causes the dispatch to be performed on the tuple space handle associated with the object `gts`, in this case the global tuple space. In order to allow formals in a template a new class Formal is introduced, and when instantiated it contains a type descriptor of the formal field in the template (in this case `integer`). In the Java embedding the `obtain` primitive returns a tuple (as in ISETL-Linda[4]). A method is provided in the Tuple class to extract a field and in line 14 this is used to extract the second field of each returned tuple in order to print it to the screen.

```
1   public class OutTest {
2     public static void main(String[] args) {
3       TupleSpace gts = new TupleSpace();
4       Tuple AllTuples[] = new Tuple[100];
5       int Handles[] = new int[100];
6       gts.tsc("GTS");
7       for (int x = 0; x < 100; x++)
8         Handles[x] = gts.dispatch(new
9                Template(new Integer(x),
10               new Formal("integer")), true);
11      for (int x = 0; x < 100; x++) {
12        AllTuples[x] = gts.obtain(Handles[x]);
13        System.out.println(x + " " + ( (Integer)
14           AllTuples[x].GetField(2)).intValue());
15    }
16  }
```

Figure 3. A Java agent which, using pipelining, retrieves 100 tuple from a tuple space.

3. Case study: The Virtual University

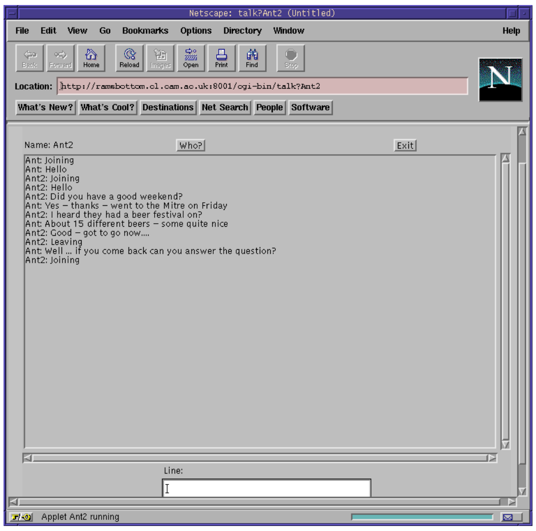
The aim of the example system is to provide the basic infrastructure of a Virtual University which supports distance learning for geographically separated students. The main method of interaction between university members will be through World Wide Web based tools which interact using the C^{2}AS. We envisage this university as having a number of students and lecturers who interact with each other even though they are not on the same campus or even in the same country or continent.

A Virtual University will provide a set of courses on different subjects. To support these courses a number of CSCW tools are required which provide the different forms of interaction that students at a physical university experience. Rather than concentrating on how course content is provided, we have concentrated on the interaction tools. The interaction tools must be able to support temporal separation so that the users can share information concurrently but also potentially share the information at different times, e.g. pupils living in different time zones.

Currently, two of the tools have been developed, a talk tool and a feedback tool. The talk tool is now described in greater detail. The talk tool is based on the talk tool described in Rowstron[10]. The tool is designed to facilitate communication between humans through time. The tool maintains a conversation which is simply the entries made. These are ordered so the most recent additions to the conversation are at the end of the conversation and this is stored in a tuple space. When a talk tool terminal is started the conversation to that point is printed, and the user can then add lines to the conversation. Because the conversation is stored in a tuple space, *all* talk tool terminals can terminate and the conversation will not be lost.

In the virtual university such a tool is used to enable inter-pupil conversation, tutorial group conversation, and individual pupil-supervisor conversations. Another tool is currently under development to allow an agent to detect when a conversation has been altered. Therefore, a supervisor can 'watch' all the unique conversations between himself and each pupil. The talk tool is also currently being extended to support the cross posting of lines, and the 'copying' of lines of text from one conversation to another, so a private conversation between a supervisor and a student could be posted to the tutorial group conversation for example.

This is all achieved using simple agents, that access the tuple spaces. There are no servers running which manage the conversations. The agents manipulate the tuples themselves. The implementation of the basic talk tool can be seen in Rowstron[10]. There are three versions of the talk tool, a C version, a Java version and a Java applet version. Figure 4 shows the screen shot of the Netscape browser running the Java Applet version of the talk tool.


Netscape: talk?Ant2 (Untitled)
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
What's New? What's Cool? Destinations Net Search People Software
Who? Exit
Line:
Applet Ant2 running

Location: http://ramsbottom.cl.cam.ac.uk:8001/cgi-bin/talk?Ant2

Name: Ant2

```
Ant: Joining
Ant: Hello
Ant2: Joining
Ant2: Hello
Ant2: Did you have a good weekend?
Ant: Yes – thanks – went to the Mitre on Friday
Ant2: I heard they had a beer festival on?
Ant: About 15 different beers – some quite nice
Ant2: Good – got to go now....
Ant2: Leaving
Ant: Well ... if you come back can you answer the question?
Ant2: Joining
```

Figure 4. Screen shot of a Netscape browser running the talk tool as a Java applet.

4. Comparison with other approaches

CORBA[15] (Common Object Request Broker Architecture) is a standardized open object-based system. It aims to support a distributed computing environment using heterogenous platforms. CORBA provides a client/server type architecture, with an Object Request Broker (ORB) providing references to server objects (which can dynamically change). Currently, the client/server approach is increasingly seen as an intermediate stage in moving from mainframe orientated applications to collaborative (peer-to-peer) computing[8].

Tuple spaces provide persistence of the information stored within them, and also allow organised distributed data structures to be created. The agents in C^{2}AS can communicate and synchronise by manipulating these structures in the tuple spaces. This removes the need to supply servers to support the storage and manipulation of such data. If we consider the talk tool described in the last Section. In a client/server model a server would have to be written which managed the conversations. However, in the case on C^{2}AS the agents can manipulate the tuples that store the conversation without requiring a server to be implemented.

Java RMI enables the creation of distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java Virtual Machines, possibly on different hosts. However, this technology assumes the homogeneous environment of the Java Virtual Machine and does not support a heterogeneous, multi-language environment. From the client's perspective, Java RMI is equivalent in behaviour to an Remote Procedure Call (RPC), i.e. the client invokes the remote method, and then blocks waiting for the response. For interactive Java based programs this could potentially be a problem, due to the communications overhead (for the same reasons the C^{2}AS uses the BONITA primitives rather than the Linda primitives). Also, as with CORBA, it is fundamentally based on a client-server model. If the talk tool was written using RMI then, again a server would be required to manage the conversation, because there is no notion of persistence in Java.

PageSpace[3, 2] is a platform which supports open distributed applications using the Web. It utilises a global tuple space. Access to the tuple space is provided by using the Linda primitives. PageSpace also supports Laura[17] which is a co-ordination language for agents in distributed systems that offer and request services using a service space. This service space can be considered as a tuple space, where the offers and requests which are tuples are inserted into the tuple space.

The PageSpace architecture is very different from the C^{2}AS architecture. The architecture is described in detail in Ciancarini[2]. The architecture uses many different categories of agent. Alpha agents interact with humans and are "located" within Web browsers as Java applets. These communicate with beta agents. There appears to be one beta agent per user who can use the system, and this agent acts as a gateway for the user into the PageSpace system. These agents then intact with other agents within the system which actually perform the computation. This architecture means that alpha agents do not perform computation, all computation is performed in the PageSpace by other agents.

In C^{2}AS the agents that run as applets in the browser can directly co-ordinate with other agents running as applets in other browsers. If a poker game was written using C^{2}AS then the agent running in the browser would co-ordinate, through tuple spaces, with the other poker agents. This potentially makes the writing of agent systems much simpler and ensures that computation is performed on the user's workstation rather than having to run a number of agents within the PageSpace system. However, this has the drawback that state is stored in the agent which under some circumstances can be a problem. If an agent has removed a tuple that other agents require and then dies then all the agents may well be unable to continue because the tuple is missing. We are currently considering the addition of transaction processing properties to the agents, thereby providing sets of atomic operations. PLinda[7] uses such a technique to provide fault tolerance.

C^{2}AS provides low-level co-ordination. PageSpace provides higher-level co-ordination by incorporating Laura. In C^{2}AS it is up to the developers of an agent system to decide the co-ordination style and protocol that they wish to use. It is possible to develop a Laura style co-ordination language on top of the basic functionality of C^{2}AS, but it is equally possible to develop other co-ordination mechanisms on top of C^{2}AS.

W3-Linda[14] used the Common Gateway Interface (CGI) to provide a method for allowing tuples to be inserted into a single global tuple space, and tuples retrieved. The CGI scripts processed information provided via a HTML FORM. A program consists of one or more CGI scripts, that insert the tuples and generate a new HTML page that is returned to the users Web browser. Due to the nature of HTTP (client initiated), once the HTML page has been returned to the user's browser any future update of that page is dependent upon the user of the browser initiating it by clicking on a button or something similar to cause an appropriate HTTP operation to be initiated with the server. This means interactive programs which update dynamically are not possible, and interaction with the tuple space is very restricted.

However, Schoenfeldinger describes a course evaluation tool which was developed using 3W-Linda. Students are presented with an questionnaire written in HTML FORM. The results of this are inserted into the tuple space using a CGI script. These are then processed by another agent who maintains the results in the tuple space. Also the agent maintains a graph. People can then ask to see the current status of the results. Our feedback tool could be used to perform such an operation. With our feedback system the result agent dynamically updates the display as new results are processed, without the user needing to request updates explicitly. Secondly, the graph generation and result formatting is performed by the result agent.

Gutfreund[6] proposed the use of Linda to allow different parts of a Web browser to communicate. Because of the spatial separation provided by tuple spaces, the functionality of the Web browser can be extended by adding a new process without the other processes being explicitly aware of it when the browser was first started. Therefore, the browser becomes a collection of interacting agents that provide the services. Since this work, main stream browsers have used the concept by providing support for helper applications.

5. Conclusion

In this paper we have presented an overview of the Cambridge Collaborative Agent System. The aim of C^{2}AS is to provide a flexible but yet simple system to allow geographically diverse agent based systems to communicate.

There are a number of important areas that the prototype system does not address which are primarily the issues of: scalability, agent migration and security. Current work on the development of tuple space management systems has shown the potential to allow the systems to be very scalable[9]. The techniques used in these kernels are based on the *dynamic* analysis of running agents tuple space usage. The information required can be gathered in an implicit manner, requiring no additional information to be added to the agent by the programmer. However, such kernels have not yet been implemented.

We believe that explicit migration operations are not required. For agents which do not interact with humans then the C^{2}AS run-time system decides when to migrate an agent and to where. One of the advantages of spatial separation is that agents are unaware of where the destination agent is located. Agents are written in a fashion that makes them completely independent of physical location. All an agent needs to be able to do is to use the tuple spaces it has access to. This means the programmer explicitly asking to migrate the agent appears rather strange.

The use of associative shared memories in the form of tuple spaces, provides a simple, efficient and potentially scalable approach to developing an infrastructure for Web based agent systems. The simplicity means that agent systems can be created quickly. The ability to store data structures in a tuple space, largely removes the need for a client-server approach to Web based systems, by allowing agents to directly access and manipulate the stored information.

6. Acknowledgements

Sheng Li and Radina Stefanova were supported by a CASE grant from the EPSRC of the UK and *The Olivetti & Oracle Research Laboratory*. Antony Rowstron was supported by a research grant from *The Olivetti & Oracle Research Laboratory*. The authors would like to thank Andy Hopper and Stuart Wray for their support.

References

[1] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.
[2] P. Ciacarini, R. Tolksdork, and F. Vitali. Weaving the Web in a PageSpace using coordination. 1996.
[3] P. Ciancarini, A. Knocke, R. Tolksdorf, and F. Vitali. PageSpace: An architecture to coordinate distributed applications on the web. In *5th International World Wide Web Conference*, 1995.
[4] A. Douglas, A. Rowstron, and A. Wood. ISETL-LINDA: Parallel programming with bags. Technical Report YCS 257, University of York, 1995.
[5] D. Gelernter. *Mirror worlds*. Oxford University Press, 1992.
[6] Y. Gutfreund. WWWinda: An orchestration service for WWW browsers and accessories. In *Electronic Proceedings 2nd International World Wide Web Conference: Mosaic and the Web*, 1994.
[7] K. Jeong and D. Shasha. Persistent Linda 2: a transaction/checkpointing approach to fault-tolerant linda. In *Proceedings of the 13th Symposium on Fault-Tolerant Distributed Systems*, 1994.
[8] T. Lewis. Where is client/server software headed? *IEEE Computer*, 28(4):49–55, 1995.
[9] A. Rowstron. *Bulk primitives in Linda run-time systems*. PhD thesis, Department of Computer Science, University of York, 1997.
[10] A. Rowstron. Using asynchronous tuple space access primitives (BONITA for process co-ordination. Accepted for Coordination'97, 1997.
[11] A. Rowstron and A. Wood. An efficient distributed tuple space implementation for networks of heterogenous workstations. Technical Report YCS 270, University of York, 1996.
[12] A. Rowstron and A. Wood. An efficient distributed tuple space implementation for networks of workstations. In *Euro-Par'96*, LNCS 1123, pages 510–513. Springer-Verlag, 1996.
[13] A. Rowstron and A. Wood. BONITA: A set of tuple space primitives for distributed coordination. In *30th Hawaii International Conference on System Sciences 1*, pages 379–388. IEEE CS Press, January 1997.
[14] W. Schoenfeldinger. WWW meets Linda. In *Electronic Proceedings 4th International World Wide Web Conference: The Web Revolution*, 1995.
[15] J. Siegel. *CORBA Fundamentals and Programming*. John Wiley & Sons Inc., 1996.
[16] V. Sunderam, J. Dongarra, A. Geist, and R. Manchek. The PVM concurrent computing system: Evolution, Experiences, and Trends. *Parallel Computing*, 20(4):531–547, 1994.
[17] R. Tolksdorf. Coordinating services in open distributed systems with LAURA. In *Proceedings of Coordination '96*, LNCS 1061, pages 386–402. Springer-Verlag, 1996.