Jada: a coordination toolkit for Java

P. Ciancarini    D. Rossi

Technical Report UBLCS-96-15
October 1996

Department of Computer Science
University of Bologna
Piazza di Porta S. Donato, 5
40127 Bologna (Italy)
Recent Titles from the UBLCS Technical Report Series

95-13 Optimal Multi-Block Read Schedules for Partitioned Signature Files, P. Ciaccia, August 1995.
96-5 The Shape of Shade: a Coordination System, S. Castellani, P. Ciancarini, D. Rossi, March 1996.
96-7 Using Bayesian Belief Networks for the Automated Assessment of Students’ Knowledge of Geometry Problem Solving Procedures, M. Roccetti, P. Salomoni, March 1996.
96-9 Towards an Algebra of Actors, M. Gaspari, April 1996.
96-10 Mobile Petri Nets, A. Asperti, N. Busi, May 1996.
96-12 A Logic Coordination Language Based on the Chemical Metaphor, P. Ciancarini, D. Fogli, M. Gaspari, July 1996.
Jada: a coordination toolkit for Java

P. Ciancarini  D. Rossi

Technical Report UBLCS-96-15

October 1996

Abstract

Java is a concurrent, multithreaded language, but it offers little help for distributed programming. Thus, we introduce Jada, a coordination toolkit for Java. Coordination among either concurrent threads or distributed Java objects is achieved via shared tuple spaces. By exchanging tuples through tuple spaces, Java programs and applets can exchange data or synchronize their actions over the Internet, a LAN, a single host, or even inside a browser enabled for Java multithreading.

The access to a tuple space is performed using a set of methods of a tuple space object. Such operations are out (to put a tuple in the tuple space), in and read (to get or to read associatively a tuple from the tuple space), and others, mostly inspired by the Linda language.

Jada does not extend the syntax of Java because it is a set of classes. We show how it changes the way we design multiuser, distributed applications based on the WWW.
1 Introduction

One key to the rapid diffusion of the World Wide Web is its simplicity. This suggests that any enhancement to WWW middleware should be simple as well.

The main components of WWW’s success are currently:

1. The HTTP, a communication protocol for interaction between a Web client and a Web server. HTTP is a minimal, asynchronous, stateless protocol for the rapid retrieval of remote resources. HTTP features are the statelessness of the connections (the context must be re-established for every connection) and the minimality of the dialog (each connection consists of one request and one response only: for instance, since images are stored in resources external to text data, in order to fetch a document containing five images six connections are necessary).

2. The URIs, a naming scheme providing unique global identification of network resources. URLs are a subset of URIs providing precise information on the correct place to address and the protocol to use when accessing them. In fact, URLs allow clients to request data to several kinds of servers, beyond HTTP: e.g. SMTP (e-mail), NNTP (news), FTP, etc., with a simplified and unified access syntax.

3. The HTML, a formatting language for specifying Web pages to be displayed by a Web client.


Using Java for building multiuser systems, the WWW can be seen as a platform that aims at integrating Internet-based services and applications. Coordination applications in which proactive or reactive processing is required, like groupware or workflow, are especially suited to be at least monitored through the Web, and in several cases they can be fully controlled. However, several deficiencies, such as ad-hoc technical solutions modifying the basic protocols, do prevent the current WWW from satisfying the needs for designing, implementing, and establishing applications in which distributed processing is required.

In this paper we concentrate on the Java component. We are interested in using WWW+Java as a platform for distributed multiuser applications, e.g. groupware and electronic commerce. Java seems quite promising for this kind of applications, because it offers a uniform platform to coordinate software running on heterogeneous architectures.

Although Java is multithreaded, it is not a language for distributed programming. Typically it must be integrated by sockets libraries to support distributed processing. However, programming distributed systems using sockets is notoriously boring, difficult and error-prone. An alternative strategy consists of adding to Java high-level coordination mechanisms: this is the theme of this paper.

This report has the following structure: in Sect.2 we shortly recall Java; in Sect.3 we introduce Jada (Java+Linda); in Sect.4 we discuss how we implemented Jada. In Sect.5 we show some simple Jada programs. In Sect.6 we compare our system with other similar systems; Sect.7 concludes the report. Two appendices complete the paper: in Sect.8 we describe the Jada API; in Sect.9 we give some hints on how to run Jada applications.

2 Java

When the World Wide Web captured the interest of the industrial world, many companies started projects to extend WWW capabilities in order to make it a platform for multiuser applications for users either concentrated in a local area network (intranet) or geographically distributed over the Internet.

Some software engineers working at Sun Microsystems realized they already had a tool other companies were looking for. In fact, at the beginning of 90s a team of engineers had started a project aimed to develop a programming language and environment suitable for the consumer-electronics market. They started using C++, but soon they realized that some requirements of robustness and simplicity, and the wide variety of architectures to be covered, introduced
problems that could be solved only by developing a new language, Java, simpler and cleaner
than C++.

In fact, Java is an object oriented language whose syntactical structure resembles C++. However, it inherits a better object-oriented semantics from SmallTalk and Objective C [AG96].

The language model is somehow limited but its strongest point is its great simplicity.

Java programs are made of objects. Java objects are instances of classes. A class can inherit
from a base class (just one, multiple inheritance is not supported) to extend the behavior of that
base class.

Java programs are compiled into a binary format [Gos95] that can be executed on many plat-
forms without recompilation using a “bytecode interpreter”, also known as Java Virtual Machine
(JavaVM). The bytecode is linked on the fly when loaded to be run [Gos95]. All the dependencies
in the byte-code are symbolic and are resolved only at load time. Thus, a Java programmer can
change the behavior of a base class without needing to recompile a derived one. Furthermore,
loading a class happens only when it is required and can happen also from a remote host in the
network. Thus, the JavaVM contains mechanisms to check the code it has to run, in order to avoid
viruses and security violations.

Java is architecture neutral: every machine-dependent code is not part of the language itself
but of its standard library. Java libraries are collections of classes called “packages”. We can think
of Java as an environment built by the language itself and its standard packages. Several aspects
of the language depend indeed from the standard packages as defined by Java designers. For
instance, multithreading is native in Java, but there is no way to create a thread other than using
a method of the Thread class.

The standard packages included in any JavaVM are:

- java.lang this package is automatically imported when compiling. It contains the defini-
tions for the root class Object and for the meta-class Class, plus threads, exceptions,
etc...
- java.io contains the classes for accessing files.
- java.net contains the classes for accessing the network.
- java.util contains utility classes such as containers, date and time managers, and so on.
- java.awt contains the classes for the Abstract Windowing Toolkit, which defines basic
graphics interface components in a form that is independent from the actual windowing
library the implementation uses (eg. Windows API for Windows and Motif for Unix).

What follows is a simple example of distributed Java program: two programs communicate
remotely exchanging “ping” and “pong” messages.

// this code describes two programs which communicate through sockets.
// Note that PING and PONG are not symmetrical

// PING

import java.io.*;
import java.net.*;

public class Ping {
    static final int port=2345;

    public void run() throws IOException {
        // listen for a connection request
        ServerSocket ss=new ServerSocket(port);
        // accept the connection and create associated streams
        Socket s=ss.accept();
        InputStream is=s.getInputStream();
        OutputStream os=new s.getOutputStream();
        // do the ping/pong
        do {
            os.write(0); // value 0 means "ping"
2.1 Java and the Internet

HotJava, a WWW browser, has been the first Internet application of Java. What made it different from other browsers is its ability to run Java code “embedded” within HTML documents. This way HTML documents become “active”, namely it is easy to put animation in HTML pages. Moreover, the same mechanism can be used to extend a browser in a natural way, implementing, for example, editors, spreadsheet, and groupware applications made of distributed objects.

The most important consequence of having Java-enabled browsers like HotJava is that a Java application can be “spread” around the network. The simplest way to take advantage of this
feature is to think about a Java application just like a document. Watching a Java application means let it run in your host, possibly integrated within your browser. The Java way to obtain this is to build each piece of Java code to be run in a browser as an “applet”.

An applet is the byte-code result of a compilation of a class that extends the class Applet contained in the java.applet package. This means that each applet inherits a standard behaviour from Applet and has to comply with a set of conventions which let them run within a Java-compatible browser.

What follows is again a ping/pong application, which include the same code already seen before. However, this time these are applets and they need a “bridge” server to maintain a message buffer through the connectionless HTTP protocol.

```java
//--BRIDGE--
import java.io.*;
import java.net.*;
public class Bridge implements Runnable {
    static final int port=2345;
    Socket s1=null, s2=null;
    // forward the incoming data to outgoing connection
    public void run() {
        Socket s=null;
        String my_name=Thread.currentThread().getName(); // who am I?
        if(my_name.equals("s1")) { // I have to listen to s1
            s=s1;
        } else { // I have to listen to s2
            s=s2;
        }
        try {
            InputStream is=s.getInputStream();
            OutputStream os=s.getOutputStream();
            while(true) {
                os.write(is.read());
            }
        } catch (IOException e) { // stop this thread on any error condition
            Thread.currentThread().stop();
        }
    }
    public static void main(String args[]) {
        try {
            ServerSocket ss=new ServerSocket(port);
            s1=ss.accept();
            new Thread(new Bridge, "s1").start();
            s2=ss.accept();
            new Thread(new Bridge, "s2").start();
        } catch (IOException e) {
            System.out.println(e);  
        }
    }
}```
Java suffers the lack of mechanisms for managing coordination among objects. There is no other way to handle a simple client/server connection than explicitly use sockets and decode byte-stream requests and replies. Things get even worse when we have the need for the objects to cooperate in a more complex fashion. If the WWW has to become an infrastructure for both developing and implementing business applications in distributed and collaborative business environments we need the ability to manage coordination between remote Java applications.

3 A toolkit for programming distributed Java applications

We solve the coordination problem for Java using a well known approach, namely adding to Java a minimal set of coordination primitives: in particular, we add operations to access Linda-like multiple tuple spaces [CG92, Cia94].

We designed Jada aiming to simplicity rather than performance. Jada, like Linda, is a minimalist coordination language. Differently from other Linda-like implementations, which usually include a preprocessor necessary because Linda slightly changes (i.e. it constrains) the host language syntax, Jada is based on a set of classes to be used to access a tuple space containing objects which are tuples, allowing the users to keep their standard Java development tools.

The main objects used in Jada are Tuple and TupleSpace. Moreover, Jada programs can use either TupleClient or TupleServer classes, or both.

3.1 Tuples

In Jada a tuple is a set of objects (also referred as items) and it is represented by the tuple.Tuple class.

This is an example of Jada tuple:

```java
Tuple my_tuple=new Tuple(new Integer(10),"test");
```

Such a tuple includes two items (we say that its cardinality is two); the first item is an Integer object, the second one is a String object. The value of the first object is 10, the value of the second object is test. An item in a tuple which has a value is said to be actual. When an item has no value it is said to be formal.

To build a tuple with one formal field of type String we can write:

```java
Tuple formal_tuple=new Tuple(new String().getClass());
```

Thus, a Class type used as argument in the constructor of a tuple is used to represent a formal field. Tuples that contain formal field are referred as formal tuples.

3.2 Tuple Spaces

Now that we know how to create tuples we can interact with a tuple space. A tuple space is a collection of tuples. To build a tuple space using Jada we write:

```java
tuple_space=new TupleSpace();
```

We can share this object between multiple threads. When we want a thread to put a tuple into the tuple space we write:

```java
tuple_space.out(new Tuple(new Integer(10),"test"));
```

This way a new tuple with two actual field will be put into the tuple space. This tuple can be later read by another (or even by the same) thread using the read or the in methods (the difference between read and in is that in removes the returned tuple from the tuple space):

```java
Tuple read_tuple=tuple_space.read(new Tuple(new Integer(10),"test"));
```

This is because the tuple used as argument for the read method is equal to the tuple we put before in the tuple space. The access to the tuple space performed by the in and read operations
is associative: these operations return from the tuple space a tuple that matches the tuple used as argument.

The matching principle is quite simple: two tuples \( a \) and \( b \) are matching if they have the same cardinality and each item of \( a \) matches the corresponding item of \( b \). Two items \( c \) and \( d \) are matching if they have the same type and (if both are actual) the same value.

Thus, the tuple

\[
\text{Tuple } a = \text{new Tuple(new Integer(10),"test"};
\]

matches the tuple:

\[
\text{Tuple } b = \text{new Tuple(new Integer(10),new String().getClass());}
\]

Note that to exchange a tuple two threads should not perform a pair of out and read operations at the same time (Jada does not support rendez-vous communication). In fact, suppose the threads \( ta \) and \( tb \) want to exchange a message: \( ta \) will put a message inside the tuple space, \( tb \) will read the message from the tuple space. If \( ta \) performs the out operation before \( tb \) performs the read operation it does not have to wait for \( tb \): it simply continues its execution, the tuple is now stored into the tuple space. When \( tb \) performs the read operation it will be able to read it.

Instead, if \( tb \) performs the read operation before \( ta \) performs the out operation, \( tb \) will be blocked until tuple that satisfy the read request will become available (i.e. until \( ta \) performs the out operation).

The in and read methods are indeed blocking. If you want the thread not to be blocked when a matching tuple for the in and read operations is not available you can use the in_nb and read_nb methods: they access the tuple space the same way as in and read but if no matching tuple is available they simply return null.

3.3 TupleServer and TupleClient

In the design of Jada we had to cope with Java way to access the network. We used a simple client/server architecture to manage the tuple spaces. To allow remote access to a tuple space, the TupleServer and TupleClient classes are provided. In fact, each tuple space is a shared remote resource accessed through a tuple space server.

The tuple space server is addressed using the IP address of the host it runs on and with its own port number (as usual with socket connections). This way we can run (almost) as many tuple space servers as we like in a network, so that applications can independently operate on several, distributed tuple spaces. TupleServer is a multithreaded server class which translates requests received from the TupleClient class in calls to the methods of the TupleSpace class.

In fact, both TupleServer and TupleClient are based on TupleSpace.

 TupleServer and TupleClient communicate using sockets.

 TupleServer uses TupleSpace to perform the requested operations.

The TupleClient class extends TupleSpace changing its internals but keeping its interface and behavior (apart from some new constructor). Thus, a TupleClient object is used just like a TupleSpace one, except that it provides access to a remote tuple space which can run in any host of the network.

What TupleClient does is to interface with a remote TupleServer object (which holds the real tuple space) and requests it to perform the in, read and out operations and (eventually) to return the result.

TupleClient needs to know where in the net the TupleServer is located and on which port it listens for requests: a set of constructors is provided to specify TupleServer host and port.

4 Some details on the implementation

Jada is implemented as a set of classes that allow either Java threads or Java applications to access associatively a shared tuple space using a small set of Linda-like operations.
A tuple space can be either local to a browser (tuple.TupleSpace, to be shared between threads) or remote (client.tupleClient, to be shared between applications) in order to build a distributed application.

In the latter case a tuple space server (server.TupleServer) must be running in a host of the network.

In Fig.1 we graphically outline how an application can access a remote tuple space.

4.1 Tuple

The basic object in implementing Jada is Tuple, that is a generic object container which can contain up to ten ordered items. We also provided lots of different constructors to build a tuple, which can also be built setting one by one its fields.

We can use a constructor to create a tuple with a contents:

```java
Tuple tuple=new Tuple("foo", new Integer(10));
```

but we can also do that incrementally using the appropriate methods:

```java
Tuple tuple=new Tuple(2);
tuple.setItem(0, "foo");
tuple.setItem(1, new Integer(10));
```

there are also other useful methods to access tuples’ fields.

Since Tuple is the basic object we use to access a tuple space, we also need:
- a method to test if two tuples match, used for associative in and read calls (matchesItem).
- a method to dump the tuple contents to a byte stream, in order to be able to send a tuple across a network or to save its state to a file (dumpItem).
- a method to restore a tuple given a set of bytes read from a byte stream (buildItem).

All these methods are also required by tuples’ items (we indeed need to compare tuples’ fields to know whether they match, and we also need to know how to dump and restore them in order to dump/restore a tuple). For this reason these methods are part of the TupleItem interface which must be implemented by any object we want to use as tuple item (the only exceptions are Integer and String which are managed as special cases).

The Tuple class itself implements the TupleItem interface so we can use a Tuple object as a field of a tuple (so that, from Jada point of view, Tuples are first class objects).

There are also no technical reasons for which we decided to use tuples and not directly objects with an interface like TupleItem (we have indeed all we need for an associative access); we decided to use tuples just for simplicity (the handling of Integer and String is then simplified) and to let Linda users feel home.

4.2 TupleSpace

A TupleSpace object in Jada is a tuple container which offers a set of thread-safe access methods. Thread-safe means that accessing a TupleSpace from different threads at the same time is safe since monitors are used to handle requests avoiding problems with critical regions.

The methods are the usual in, read and out (along with their multiple and non-blocking variants). All these methods are actually just wrappers for the doInRead and doOut methods, which are the real “engines” for this class. All the synchronization and repository management code is part of these methods. This allows to easily redefine TupleSpace behaviour by extending its class just like we did with TupleClient. For example the doOut method of TupleSpace takes care of putting the specified tuple in the tuple space or use it to reply pending in or read requests. The TupleClient’s version of doOut, instead, sends the tuples to a tuple server and asks it to manage the storing/replying. The same applies for TupleClient’s version of doInRead. We need indeed just to change these two methods to deal with a remote tuple manager (which is a TupleServer object) and use socket connections to talk with it.
Figure 1. Client-server relationships in Jada
4.3 TupleServer

As stated above a TupleServer object is used by TupleClient objects to access a (possibly remote) tuple space. A TupleServer has a TupleSpace object in it which is used to manage the tuple space. Each time a TupleClient performs a request the TupleServer runs a thread to perform the requested operation. From the architectural point of view we can think about TupleServer/TupleClient like a stub/proxy system.

If we analyze the behavior of a Jada program we can distinguish two main cases:

- the use of a TupleSpace shared among concurrent threads belonging to the same application, a situation symbolically depicted in 2.
  Each shape in such a picture represents a thread running within an application. Thread1, Thread2, and Thread3 are interacting with the tuple space (using TupleSpace methods). The TupleSpace object manages their access and takes care of handling critical regions managing. Note that some of the thread may have blocked calling a method to perform a blocking in or read operation.

- the use of a TupleSpace shared among concurrent threads belonging to different applications. In this case we have to use a TupleServer-TupleClient architecture to access the shared space, as we shown in 3.
  Thread1 and Thread2 are now part of an application while Thread3 is part of a different one. The former are using the same TupleClient object to access the tuple space, while the latter uses a private one. When a thread performs a request calling a method of the TupleClient object, the request if forwarded to the TupleServer object, using a network connection, which runs a thread to handle it. Each thread run by the tuple server then corresponds to a remote thread which performed a request.

So we can see how the same situation as before is now automatically replicated inside the application which runs the TupleServer, giving evidence to the implicit scalability of Jada’s architecture.
Figure 3. Remote threads accessing a tuple space
5 Programming with Jada

Using Jada to allow threads to exchange data is quite an easy task but we can do even better. We can use Jada to coordinate threads, thus easily solving many well-known coordination problems:

- server/client model: a server read requests in form of tuples from the tuple space. A client performs a request putting a tuple in the tuple space.
- master/worker model: the master put jobs in form of tuples in the tuple space, then reads results from the tuple space. Workers read jobs from the tuple space, execute their task, and then put the result into the tuple space.
- producer/consumer model: we decouple a producer and a consumer using the tuple space as temporary repository for data to be exchanged.
- mutual exclusion: we can use a tuple as a "token" to enter a critical section: any thread need to get such a token to enter the section. Upon exiting the section the thread releases the token (i.e. the token tuple can be put back into the tuple space)
- message passing, either synchronous or asynchronous: a sender puts the message into the tuple space (and in the synchronous case waits for an ack tuple). The receiver reads the message from the tuple space (and in the synchronous case puts an ack tuple in the tuple space).

Many other concurrency problems can be easily solved using shared tuple spaces [CG90].

We now use Jada to write a real application. Our application is again a ping/pong program. Even if it is very simple, still it is quite useful to explain how we use Jada to coordinate applications and/or threads.

```
//--PING--
import jada.tuple.*;
import java.client.*;

public class Ping {
    static final String ts_host="foo.bar";

    public void run() {
        //a tuple client interacts with the remote tuple server
        TupleClient ts=new TupleClient(ts_host);
        //do the ping/pong
        while(true) {
            ts.out(new Tuple("ping"));
            Tuple tuple=ts.in(new Tuple("pong"));
        }
    }

    public static void main(String args[]) {
        Ping ping=new Ping();
        ping.run();
    }
}

//--PONG--
```
import jada.tuple.*/
import java.client.*/

public class Pong {
    static final String ts_host="foo.bar";

    public void run() {
        // a tuple client interacts with a remote tuple server
        TupleClient ts=new TupleClient(ts_host);
        // do pingpong
        while(true) {
            ts.out(new Tuple("pong"));
            Tuple tuple=ts.in(new Tuple("ping"));
        }
    }

    public static void main(String args[]) {
        Pong pong=new Pong();
        pong.run();
    }
}

5.1 Applications
We built a number of coordination applications with Jada. For each application we also wrote an applet version, testing the expressiveness of Jada for developing distributed, cooperative applications over the Internet.

- **PingPong** a very simple application to test Jada: two processes alternatively get and put a tuple incrementing a counter. The tuple space used for coordination can be either local or remote.
- **JadaChat** a simple IRC like application. Users join the channel they prefer and share messages.
- **JadaHearts** a multiuser card game in applet form. Users can play remotely over the Internet without the need of an ad hoc graphical client, simply getting a document with their Java enabled browser (the interface is depicted in Fig.4).

We are using Jada as coordination kernel for implementing more complex Internet languages and architectures. In particular, we are developing on top of Jada the Shade/Java coordination language [CCR96], and PageSpace [CKTV96], a coordination architecture for using the WWW as middleware for building cooperative applications (eg. groupware, electronic commerce) over the Internet.

6 Related work
Jada has been developed in the context of the PageSpace project [CKTV96]. The main idea at the basis of PageSpace is to exploit coordination technology to enhance the WWW middleware. Other projects pursue a similar goal with different approaches.

In fact, although it explicitly refers to Linda, the WWWinda approach [GN94] is quite different from what we have described here. Since the integration between WWW browsers and their helper applications is extremely rudimental (it is only possible to activate the external application and to deliver the data to be displayed), the WWWinda research team designed a flexible, modular WWW browser architecture based on the Linda programming language. Several independent tools, each implementing a different part of the whole WWW browser, are activated according to needs, sharing screen space, HTTP responses, and user interaction. This allows for a highly modular architecture, where new services and tools can be added without modifications in a homogeneous framework. In order to allow cooperation among helper modules, these all
Figure 4. The JadaHearts Interface
make use of a shared tuple space. Tuples allow to upgrade the simple “activate with these
data” paradigm of browsers’ helper applications to a more complete coordination protocol. For
instance, WWWinda has been used to coordinate a distributed digital orchestra, in which several
browsers simulating musical instruments (possibly running on different machines) extract from
the shared tuple space the tune to be played note by note. No instrument is aware of how many
other instruments are present, and new ones can be added on the fly, even in the middle of a
performance.

All features of WWWinda, that we define a “client-side coordination toolkit” are easily imple-
mentable in Jada.

Instead, the WU Linda Toolkit [Sch95] is a simple interface between a WWW browser and a
WWW server implementing a tuple space service. Access to the shared tuple space is provided to
clients: users can fill in an HTML form with the appropriate Linda command to interact with the
tuple space. The main application on show is a disc-load viewer that allows a first glance check
of current disk usage of the computers of a cluster of workstations. Each workstation hosts a WU
Linda client which posts tuples in the shared tuple space describing the current load of the disks
it controls. These tuples are then collected and rendered in a user-friendly way via HTML to any
browser querying the application.

Again, all features of WU Linda, that we define a “server-side coordination toolkit” are easily
implementable in Jada.

Jada applets are very similar to Oblets, or “distributed active objects” [BN96]. An oblet is a
program written in Obliq and executed on a browser. Each oblet can use high-level primitives to
communicate with other oblets running on possibly remote browsers.

The Bauhaus “Turingware Web” designed in Yale (the homeland of Linda) is similar at least in
spirit to the WU Linda toolkit [Hup96]. The main idea consists of using a standard browser
to access a Bauhaus server. Bauhaus is a coordination language based on nested multiple tuple
spaces (multisets) which can be used in this case for both controlling the hierachical structure of
the pages of a web site, and for associating agents and their activities to the pages themselves.
For instance, one attribute of a page could be the list of users “acting” in such a page, who are
displayed by a graphic icon and can interact using some ad-hoc cooperation services.

This application of coordination technology to the WWW is based on a language richer
than Linda. We are also investigating similar extensions, based on multiple tuple spaces, in a
coordination language called Shade, whose kernel is currently written in Jada [CCR96].

7 Conclusion

Jada provides:

- coordination inside the Object Oriented Programming framework: no syntax extension for
  Java, just a set of classes. Each data type used by Jada is a Java object.
- dynamic tuple creation: being an object, a tuple can be created with new and many different
  constructors are provided in order to build a tuple from a string, an array of object or,
  simply, from a set of arguments (as seen before).
- multithreading support: Jada is multithreading aware: different threads can access the
  same tuple space; blocking request are managed at thread-level.
- open systems support: at any time threads or applications can perform operation on a tuple
  space.
- multiple tuple associative access: we provided modified flavors for the in and read
  requests in order to allow the use of multiple matching tuples.

We are using Jada as kernel for building more complex Internet languages.

Acknowledgements. We thank Robert Tolksdorf for discussions and suggestions. We thank for
support the EU Open LTR project Pagespace n. 20179. We have been partially supported also by
Italian CNR - “Progetto Coordinato Programmazione Logica”.

UBLCS-96-15 15
References


We describe here the Jada API.
It is available at www.cs.unibo.it/rossi/jada.

8.1 Package jada.tuple

Interfaces
  TupleItem
  TupleNetConst

Classes
  Tuple
  TupleSpace
  TupleString

8.1.1 Interface jada.tuple.TupleItem

public interface TupleItem extends Object

  Base TupleItem class. If you want to use a class as a tuple item you have to implement this interface in your class. The methods dumpItem() and buildItem() are really needed if you plan to use a TupleClient as tuple space.

  Remark: a class that implements TupleItem MUST have a default constructor; a class that implements TupleItem MUST be public.

Method Index
- buildItem(byte[]): Builds an item from a byte array representation.
- cloneItem(Object): Clone this item from a “source” one.
- dumpItem(): Returns a byte array representation for this item; if it is not possible for the item to do that the method returns null.
- matchesItem(Object): Returns true if this item matches the given one.

Methods
- dumpItem
  public abstract byte[] dumpItem()

  Returns a byte array representation for this item; if it is not possible for the item to do that the method returns null.

- buildItem
  public abstract boolean buildItem(byte b[])

  Builds an item from a byte array representation.

- cloneItem
  public abstract void cloneItem(Object source)

  Clone this item from a “source” one.

- matchesItem
  public abstract boolean matchesItem(Object match)

  Returns true if this item matches the given one.
public interface TupleNetConst extends Object

This interface contains a set of shared constants for the Jada package. Any class that wants to access these constants must implement this interface.

- DEFAULT_PORT The default port.
  public final static int DEFAULT_PORT
- IN_OPCODE The server requests opcode.
  public final static int IN_OPCODE
- READ_OPCODE
  public final static int READ_OPCODE
- OUT_OPCODE
  public final static int OUT_OPCODE
- PRINT_OPCODE
  public final static int PRINT_OPCODE
- RESET_OPCODE
  public final static int RESET_OPCODE
- END_OPCODE
  public final static int END_OPCODE
- MULTI_FLAG
  public final static int MULTI_FLAG
- NB_FLAG
  public final static int NB_FLAG
8.1.3 Class jada.tuple.Tuple

```
java.lang.Object
 | 
|----- jada.tuple.Tuple

public class Tuple extends Object implements TupleItem

Base tuple class.
A tuple is a ordered set of items. Each item of a Jada tuple can be either an Integer, a String, a
class that implements TupleItem (and we say it is "actual"), or a Class (and we say it is "formal").
A Jada tuple can have as much as 10 items; hence, 10 constructors are provided in order to
simplify the creation of a new tuple.

Matching rules
Tuples are used by the tuple space operations in an associative way using a matching operation.
Two tuples are matching if they have the same number of items and the i-th field of the first
tuple matches the i-th field of the second one for each i.
Two items are matching if:
- they are Integers and they represent the same value
- they are String and they have the same contents
- they implement TupleItem and the matchItem method returns true
- they are Class and they represent the same class
- one is a Class and the other is an instance of that class so the tuple

Tuple ta = new Tuple(new Integer(3), "three")

will match the tuple

Tuple tb = new Tuple(new Integer(3),new String().getClass())

Note that Tuple implements TupleItem so you can use a tuple as an item of another tuple. A
tuple can be converted to a string using toString(). A tuple string format is a list of items
between brackets separated by commas.
Items can be actual or formal. If an item is formal it starts with a ? followed by the name of
the class it represents ("i" and "s" are used as short formats for Integer and String, respectively).
If it is actual the toString() method is used to represent its value.

Note: if you do not follow the rules for writing a TupleItem, some method of this class
(especially buildItem) may generate a class-related exception which is caught and is printed to
the System.err stream.

Constructor index
- Tuple()
  Constructor: it builds an empty tuple.
- Tuple(Tuple)
  Constructor: it builds a tuple copying the given one.
- Tuple(int)
  Constructor: it builds the tuple calling init(int).
- Tuple(byte[])
  Constructor: it builds the tuple from a byte array calling buildItem.
- Tuple(Object)
  Constructor: it builds a tuple with the item item. This is the first of a set of 10 constructors
  that differ on the items number.
  Tuple(Object, Object)
  Tuple(Object, Object, Object)
  Tuple(Object, Object, Object, Object)
  Tuple(Object, Object, Object, Object, Object)
  Tuple(Object, Object, Object, Object, Object, Object)
```

UBLCS-96-15
Method Index

- \( I() \)
  Short form for IntegerClass.
- \( \text{IntegerClass()} \)
  Returns the Class for an Integer object.
- \( S() \)
  Short form for StringClass.
- \( \text{StringClass()} \)
  Returns the Class for a String object.
- \( \text{buildItem(byte[])} \)
  From the TupleItem interface: build this tuple from a tuple array.
- \( \text{buildItem(byte[], int)} \)
  Builds this tuple from a tuple array starting at position from
- \( \text{cloneItem(Object)} \)
  From the TupleItem interface: clone a tuple given a "source" one
- \( \text{dumpItem()} \)
  From the TupleItem interface: it dumps this tuple to an array of bytes
- \( \text{formal()} \)
  Returns the Class for a tuple, useful when we want a tuple as a formal field for another tuple.
- \( \text{getIndex()} \)
  Returns the index of this tuple if it is a result for a multiple in or read operation.
- \( \text{getItem(int)} \)
  Returns the Object at the given position.
- \( \text{getSize()} \)
  Returns the size (in bytes) of this object as used in the dumpItem method (Note: this method actually calls dumpItem to calculate the size if it has not already been called and this may be time-consuming).
- \( \text{initFromString(String)} \)
  Allocates basic data and sets the tuple reading its regular string representation.
- \( \text{matchesItem(Object)} \)
  Checks for the given tuple to match the current one.
- \( \text{read(InputStream)} \)
  Read this tuple from the given InputStream.
- \( \text{setItem(int, Object)} \)
  Sets an item of the tuple.
  Watch out: just String, Integer and TupleItem are valid object types.
- \( \text{toString()} \)
  Overrides Object.toString().
- \( \text{write(OutputStream)} \)
  Write this tuple to the given OutputStream.

Constructors

- \( \text{Tuple} \)
  Constructor Tuple()
  Constructor: builds an empty tuple.
• Tuple
  public Tuple(Tuple source)
  Constructor: builds a tuple copying the given one.

• Tuple
  public Tuple(int items)
  Constructor: builds the tuple calling init(int). The resulting tuple contains formal integer values.
  See also: init.

• Tuple
  public Tuple(byte bytes[])
  Constructor: builds the tuple from a byte array calling buildItem.
  See also: buildItem.

• Tuple
  public Tuple(Object item)
  Constructor: builds a tuple with the item item. This is the first of a set of 10 constructors that differ on the items number.

• Tuple
  public Tuple(Object item1, Object item2)

• Tuple
  public Tuple(Object item1, Object item2, Object item3)

• Tuple
  public Tuple(Object item1, Object item2, Object item3, Object item4)

• Tuple
  public Tuple(Object item1, Object item2, Object item3, Object item4, Object item5)

• Tuple
  public Tuple(Object item1, Object item2, Object item3, Object item4, Object item5, Object item6)

• Tuple
  public Tuple(Object item1, Object item2, Object item3, Object item4, Object item5, Object item6, Object item7)

• Tuple
  public Tuple(Object item1, Object item2, Object item3, Object item4, Object item5, Object item6, Object item7, Object item8)

• Tuple
  public Tuple(Object item1, Object item2, Object item3, Object item4, Object item5, Object item6, Object item7, Object item8, Object item9)

• Tuple
  public Tuple(Object item1, Object item2, Object item3, Object item4, Object item5, Object item6, Object item7, Object item8, Object item9, Object item10)
Methods

- **getSize**
  ```java
class getSize()
Returns the size (in bytes) of this object as used in the dumpItem method (Note: this method
actually calls dumpItem to calculate the size if it has not already been called and this may
be time-consuming).
See also dumpItem.
- **getIndex**
  ```java
class getIndex()
Returns the index of this tuple if it is a result for a multiple in or a multiple read operation.
- **IntegerClass**
  ```java
class IntegerClass()
Returns the Class for an Integer object. Useful for a Integer formal tuple item.
See also: I.
- **StringClass**
  ```java
class StringClass()
Returns the Class for a String object. Useful for a String formal tuple item.
See also: S.
- **I**
  ```java
class I()
Short form for IntegerClass.
See also: IntegerClass.
- **S**
  ```java
class S()
Short form for StringClass. See also: StringClass.
- **formal**
  ```java
class formal()
Returns the Class for a tuple; useful when we want a tuple as a formal field for another
tuple.
- **initFromString**
  ```java
protected void initFromString(String s)
Allocates basic data and sets the tuple reading its regular string representation.
- **setItem**
  ```java
public void setItem(int index, Object value)
Sets an item of the tuple.
Watch out: just String, Integer and TupleItem are valid object types.
See also: getItem.
- **getItem**
  ```java
public Object getItem(int index)
Returns the Object at the given position.
Watch out: this is not a copy: its the object itself!
See also: setItem.
• matchesItem
  ```java
  public boolean matchesItem(Object object)
  ```
  Checks if the given tuple matches the current one.

• toString
  ```java
  public String toString()
  ```
  Overrides `Object.toString()`. Uses the same representation `initFromString` uses to init
  a tuple.
  Overrides: toString in class Object.
  See also: `initFromString`.

• write
  ```java
  public void write(OutputStream os) throws IOException
  ```
  Write this tuple to the given OutputStream.

• read
  ```java
  public boolean read(InputStream is) throws IOException
  ```
  Read this tuple from the given InputStream. Return false if an "empty" tuple is read.

• cloneItem
  ```java
  public void cloneItem(Object object_source)
  ```
  From the TupleItem interface: clone a tuple given a "source" one.

• dumpItem
  ```java
  public byte[] dumpItem()
  ```
  From the TupleItem interface: dumps this tuple to an array of bytes.

• buildItem
  ```java
  public boolean buildItem(byte b[])
  ```
  From the TupleItem interface: build this tuple from a tuple array.

• buildItem
  ```java
  public boolean buildItem(byte b[], int from)
  ```
  Builds this tuple from a tuple array starting at position from.
8.1.4 Class jada.tuple.TupleSpace

java.lang.Object
+----jada.tuple.TupleSpace

public class TupleSpace extends Object

Tuple Space class. The usual Linda operations are methods of this class.

Constructor index
- TupleSpace() Constructor.

Method Index
- in(Tuple) Performs the in operation.
- in(Tuple[], int) Performs the in operation on a tuple array.
- inRead(Tuple[], int, boolean, boolean) Performs in and read operations either blocking or non-blocking from a tuple array.
- in_nb(Tuple) Performs the non-blocking inp operation.
- in_nb(Tuple[], int) Performs the non-blocking inp operation on a tuple array.
- out(Tuple) Performs the out operation.
- out(Tuple[], int) Performs the out operation on a tuple array.
- print() Prints the tuple space contents to System.out.
- read(Tuple) Performs the read operation.
- read(Tuple[], int) Performs the read operation on a tuple array.
- read_nb(Tuple) Performs the non-blocking readp operation.
- read_nb(Tuple[], int) Performs the non-blocking readp operation on a tuple array.
- reset() Resets the tuple space (by deleting its contents).

Constructors
- public TupleSpace()
  Constructor. Builds an empty tuple space.

Methods
- public synchronized void out(Tuple item)
  Performs the out operation, which puts the item into the tuple space. You should use actual tuples here; however, no check is performed for performance reasons.
- public synchronized void out(Tuple tuples[], int n_tuples)
  Performs the out operation on a tuple array. Puts n_tuples from the array tuples into the tuple space.
- public Tuple read_nb(Tuple match)
  Performs the non-blocking readp operation. Returns a tuple (if any) that matches the match tuple in the tuple space. If such a tuple does not exist returns null.
- public Tuple read_nb(Tuple match[], int n_tuples)
  Performs the non-blocking readp operation on a tuple array. Returns a tuple (if any) that matches one item of the match tuple array in the tuple space. If the tuple does not exist it returns null.
• **read**

  ```java
  public Tuple read(Tuple match)
  ```

  Performs the read operation, which returns the tuple that matches the match tuple. This method does not return until such a tuple is available.

• **read**

  ```java
  public Tuple read(Tuple match[], int n_tuples)
  ```

  Performs the read operation on a tuple array. Returns the tuple that matches one item of the match tuple array. This method does not return until such a tuple is available.

• **in**

  ```java
  public Tuple in(Tuple match)
  ```

  Performs the in operation. Returns a tuple that matches the match tuple. This method does not return until such a tuple is available.

• **in**

  ```java
  public Tuple in(Tuple match[], int n_tuples)
  ```

  Performs the in operation on a tuple array. Returns a tuple that matches one of the items of the match tuple. This method does not return until such a tuple is available.

• **inRead**

  ```java
  public synchronized Tuple inRead(Tuple match[], int n_tuples, boolean remove, boolean blocking)
  ```

  Performs in and read operations either blocking or non-blocking from a tuple array. The blocking, non-blocking, in and read methods of this class are just wrappers for this method.

• **reset**

  ```java
  public synchronized void reset()
  ```

  Resets the tuple space (by deleting its contents)

• **print**

  ```java
  public synchronized void print()
  ```

  Prints the tuple space contents to System.out.
8.1.5 Class jada.tuple.TupleString

    java.lang.Object
    \|-- jada.tuple.Tuple
       \|-- jada.tuple.TupleString

public class TupleString extends Tuple

    This is a utility class that allows to create a tuple directly from its string representation.
    Watch out: if you use this object as a tuple item be aware that TupleString is not the same as
    Tuple. That is, the tuples:
    Tuple t1=new Tuple(new TupleString("(a, b, c)");
    Tuple t2=new Tuple(new Tuple().getClass())

do not match!

Constructor index
- TupleString() Default constructor.
- TupleString(String) Constructor: builds the tuple from the string representation.

Constructors
- TupleString
    public TupleString()
    Default constructor. Builds an empty tuple.
- TupleString
    public TupleString(String s)
    Constructor: builds a tuple from its string representation.
    Watch out: no checking is performed on the string correctness.
8.2 Class jada.client.TupleClient

java.lang.Object

| +---- jada.tuple.TupleSpace
| +---- jada.client.TupleClient

public class TupleClient extends TupleSpace implements TupleNetConst

Interface to a remote tuple space server. The usual Linda operations are performed on a remote tuple space. On errors exceptions are printed and System.exit(1) is called.

Constructor index

- TupleClient(String, int) Constructor: builds a tuple client to connect a server with given hostname and port
- TupleClient() Constructor: builds a tuple client to connect a server on host localhost.
- TupleClient(String) Constructor: builds a tuple client to connect a server with given hostname

Method index

- end() Sends the end request to the remote tuple server.
- inRead(Tuple[], int, boolean, boolean) Sends the operation request to the remote tuple server. Overrides inRead in class TupleSpace.
- out(Tuple) Sends the out request to the remote tuple server.
- print() The print request.
- reset() Sends the reset request to the remote tuple server.

Constructors

- TupleClient
  public TupleClient(String _host, int _port) Constructor: builds a tuple client to connect a server with given hostname and port
- TupleClient
  public TupleClient()
  Constructor: builds a tuple client to connect a server on host localhost.
- TupleClient
  public TupleClient(String _host)
  Constructor: builds a tuple client to connect a server with given hostname

Methods

- inRead
  public Tuple inRead(Tuple match[], int _nTuples, boolean remove, boolean blocking)
  Sends the operation request to the remote tuple server. Overrides: inRead in class TupleSpace.
• **out**

  public void out(Tuple item)

  Sends the **out** request to the remote tuple server. Overrides: **out** in class TupleSpace.

• **reset**

  public void reset()

  Sends the **reset** request to the remote tuple server.
  Overrides: **reset** in class TupleSpace.

• **print**

  public void print()

  The **print** request. This is a debug request. It causes the remote tuple server to print the contents of the tuple space to its standard output.
  Overrides: **print** in class TupleSpace.

• **end**

  public void end()

  Sends the **end** request to the remote tuple server.
  Watch out: this shuts down the server.
8.3 Class jada.server.TupleServer

```java
public class TupleServer extends Object implements Runnable, TupleNetConst
```

This is the remote tuple server; it accepts socket connections and runs new threads able to handle requests.

**Constructor Index**
- **TupleServer()**
  Constructor: builds a new tuple server listening at the default port.
- **TupleServer(int)**
  Constructor: builds a new tuple server listening at a given port.

**Method index**
- **main(String[])**
  TupleServer's entry point as standalone Java program.
- **run()**
  The main loop: it accepts socket connections and runs new threads.
- **setVerbose(boolean)**
  Sets the verbose mode.
- **waitToStart()**
  This method returns when the server is up and running.

**Constructors**
- **TupleServer**
  ```java
  public TupleServer()
  Constructor: builds a new tuple server listening at the default port.
  ```
- **TupleServer**
  ```java
  public TupleServer(int new_port)
  Constructor: builds a new tuple server listening at a given port.
  ```

**Methods**
- **run**
  ```java
  public void run()
  The main loop: it accepts socket connections and runs new threads.
  ```
- **waitToStart**
  ```java
  public synchronized void waitToStart()
  This method returns when the server is up and running.
  ```
- **setVerbose**
  ```java
  public void setVerbose(boolean mode)
  Sets the verbose mode. Requests will be printed to System.out stream.
  ```
- **main**
  ```java
  public static void main(String args[]) throws IOException
  TupleServer's entry point as standalone Java program. It is used to run a remote tuple server which accepts remote connections and runs a ServerThread to listen to new sockets.
  Usage: java jada.server.TupleServer [-v] [-p port]
  The -verbose option causes debug output to be written on the terminal. The -p port option can be used to specify the connection port.
  ```
9 Appendix: Compiling and running a Jada application

To test your installation you can try to compile and run some of the programs in the demo directory.

First of all you need to set your CLASSPATH directory to the directory that contains the Jada installation directory.

Then you can compile `PingPong.java` with the following command:

```
javac PingPong.java
```

If the compilation succeeds you will have three new class files: Ping.class, Pong.class, and PingPong.class.

The `run` method of Ping class gets a tuple of the form `(ping, ?Integer)` (where `?Integer` is any object instance of the Integer class), creates a new Integer object (say, `cnt`) which is obtained adding 1 to the second field of the got tuple, puts the tuple `(pong, cnt)` back into the tuple space, and loops these three steps again.

The operations are performed on the `tuple_space` tuple space which can be either a local or a remote one.

The Pong class is specular to the Ping one (it inputs `(pong, ?Integer)` and outputs `(ping, Integer)`).

The PingPong class creates a new local tuple space and runs Ping and Pong as concurrent threads passing them the newly created tuple space to be shared.

Now run the PingPong class:

```
java PingPong
```

you will see the two Ping and Pong threads which coordinate themselves using the tuple space.

Now we can use the same classes for a distributed ping-pong application.

Let us start with just one host: we assume we can use a different terminal window for each Java program. First of all we have to run a tuple space server.

```
java jada.server.TupleServer
```

(you can use the `-v` switch to see what the tuple server is doing).

Then we have to run the Ping application:

```
java Ping
```

Then the Pong application:

```
java Pong
```

Now we switch to a really distributed environment: say we have three hosts: alpha, beta, and gamma.

We run the tuple server on the alpha host. We can run the Ping application on the beta host, but we have to tell the application where the remote server is:

```
java Ping alpha
```

The same applies for Pong on the gamma host:

```
java Pong alpha
```

We are running now a distributed ping-pong application over a network including alpha, beta, and gamma.

We can run our distributed application on the Internet using the WWW and any Java-enabled browser. In fact, we will run now the ping application in an applet form.

The PingApplet class in the PingApplet.java file is an applet that can be embedded within an HTML document (ping.html is provided as example).

Due to Java applets security restrictions we have to run the tuple space server on the same host from which the applet is loaded (this is usually the host that runs your httpd server).

Let us say this host has name `http-host`. We have to launch the server on `http-host` as follows:
java jada.server.TupleServer

Now we can run the pong application on any host (including http-host):

java Pong http-host

You can now use your favourite Java enabled browser (such as Netscape Navigator) or an appletviewer to load the Ping applet:

appletviewer http://www.your.domain/path/to/jada/demos/directory/ping.html

You should now see a scrolling list which reports the ping activity of your applet.

The ping-pong application is of course a silly example, but you can use Jada to coordinate more useful programs.