Javaspace Implementation

Dinesh D. Puri
Asst. Professor Department of Computer Engineering
SSBT College of Engineering and Technology, Bambhori, Jalgaon, Maharashtra, India

Atul V. Dusane
Asst. Professor Department of Computer Engineering
SSBT College of Engineering and Technology, Bambhori, Jalgaon, Maharashtra, India

Nitin Y. Suryawanshi
Asst. Professor Department of Computer Engineering
SSBT College of Engineering and Technology, Bambhori, Jalgaon, Maharashtra, India

Abstract- This paper discusses various coordination models and coordination languages. Difference between various coordination models is specified here. The coordination language Linda, TSpace and Javaspace are studied as coordination languages.

Linda is having some limitation as coordination language. So javaspace is good option for implementation of distributed application. But javaspace language is having limited primitives compared with another language and the primitives which are not present in javaspace are implemented as J2space.

Keywords – *Coordination language*, *shared space, Tuplespace, Javaspace, primitives, J2space.*

I. INTRODUCTION

Programming for distributed and parallel system can be seen as combination of computing part and coordination part. The computing part consists of number of process. Involved in manipulating data. Coordination part is responsible for communication and cooperation between processes. The concept of coordination is closely related to those of multilinguality and heterogeneity. The actual programming languages used to write computational code play no important role in setting up the coordination components. There are many definitions of what is coordination ranging from simple ones such as 1."Coordination is managing dependencies between activities." 2."Coordination is the additional information processing performed when multiple, connected actors pursue goals that a single author pursuing the same goals would not perform." 3."Coordination is the process of building programs by gluing together active pieces". 4."A coordination model is the glue that binds separate activities into an ensemble."[1] A coordination model can be viewed as a triple **(E, M, F)**, Where **E** represents the entities being coordinated, **M** the media used to coordinate the entities, and **F** the semantic framework the model adheres to.

A coordination language is a language defined specifically to allow two or more parties or components to communicate, to accomplish specific goal. Proper functioning of many of today's application depends on the mutual cooperation and interaction of computers and human users. Special coordination language has been designed, in which one can specify, analyze and control the cooperation between various system components.

Javaspace is a coordination tool [7]. It uses data driven coordination model. Our goal is to identify the limitation of Javaspace in terms of coordination primitives. For this purpose we can study the various coordination languages and compare language with the Javaspace. We are comparing these languages in terms of primitive set, basic entity being coordinated, mechanism of coordination and coordination medium of various coordination languages. In this paper primitives which are not present in the Javaspace are implemented. Those primitives are very important to implement the complex application.

II. CO-ORDINATION MODEL

Any coordination language belongs to coordination model. There are two types of coordination models, data-driven and control driven coordination model. There are two types of it. Data-driven and control driven model [1].

2.1 Data-driven Coordination model

In data-driven co-ordination model processes use shared data space for communication. Processes can post or broadcast information into the medium and also they can retrieve information from the medium either by actually removing this information out of the shared medium or taking a copy of it

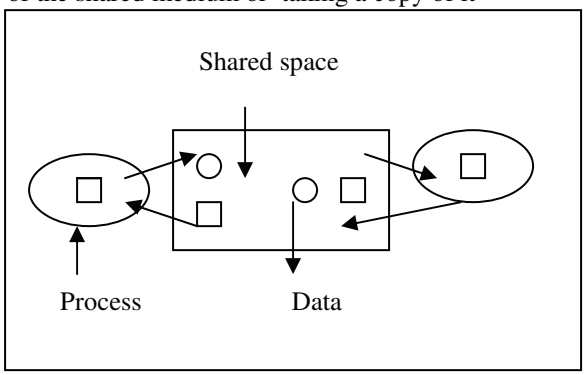

Fig. 1 Data-driven Model

Example of data-driven model is Linda, space and Javaspacs.

2.2 Control driven Coordination model

With control-driven coordination models, the state of the computation at any moment in time is defined in terms of only the coordinated patterns that the processes involved in some computation. referred to as ports

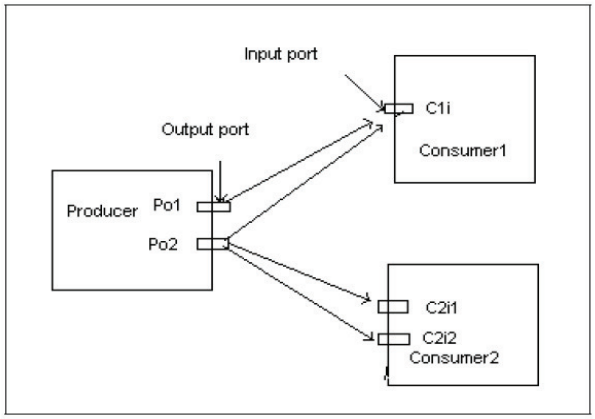

Fig. 2 Control-driven Model

The components being coordinated are considered as black boxes that produce and consume data on well-defined interfaces to the external world. Example of control-driven language is Bonita, Laura, and Sonia [1].

III. JAVASPACE: JAVA BASED CO-ORDINATION LANGUAGE

Javaspace is the coordination language which is pure java-based. Any coordination language is the mixing of co-ordination primitives with the host language. In Linda and Tspace host languages can be C, C++, Erlang, and Smalltalk. But java language is not used. Only Javaspace language can take the advantage of java language.

3.1 Primitives of Javaspace

Co-ordination language is having primitives associated with it. These primitives are mixed with the host language. In Javaspace host language is Java language

There are four primary kinds of operations that you can invoke on a Javaspaces service [7]. Each operation has parameters that are entries, including some that are templates, which are a kind of entry. Following are the primitives of Javaspace.
Write ( ): Write the given entry into this JavaSpaces service.
Read ( ): Read an entry from this JavaSpaces service that matches the given template.
Take ( ): Read an entry from this JavaSpaces service that matches the given template, removing it from this space.
Notify ( ): notify a specified object when entries that match the given template are written into this JavaSpaces service.

3.2 Comparison

In this section Javaspace is compared with Linda and Tspace languages. Advantages of Javaspace over Linda are presented in table 3.1. When Javaspace is compared with Linda, the various attributed like, multiple space, identification mechanism, matching entry with standard template or wildcards and notification mechanism is discussed.

Table 3.1 Comparison with Linda

Attribute	Javaspace	Linda
Multiple space	Yes	No
Use of Wildcards	Yes	No
Notification	Yes	No
Identification	Yes	No
Transaction	Yes	No

Tspace is having large number of primitives as compared with primitives present in Javaspace.

Table 3.2 Comparison with Tspace

Primitive	Javaspace	Tspace
Write	Yes	Yes
Read	Yes	Yes
Take	Yes	Yes
Notify	Yes	Yes
Delete	No	Yes
CountN	No	Yes
Scan	No	Yes
ConnsumScan	No	Yes

The primitives which are not present in the Javaspace can be implemented, so that primitive set of Javaspace will be enriched.

IV. EXTENDING JAVASPACE AS J2SPACE

The Javaspace was having very limited primitive set. As we have compared Javaspace with the Tspace and came to know that some primitives are not present in the Javaspace.

4.1 Primitive implementation

If we implement above primitives, primitive set of Javaspace will increase so we can call this new modified language as “J2Space” which is extending Javaspace. Class diagram of implementation of primitives is as follows.

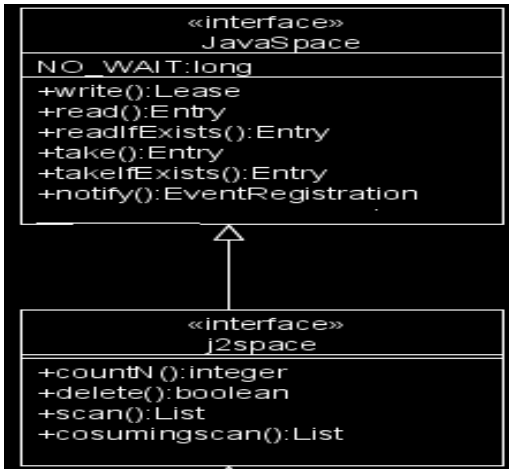

Fig. 3 J2Space Class diagram

For implementation of J2space primitives Jini technology is used[4]. Jini technology is service oriented architecture that defines a programming model which extends java technology to enable construction of secure distributed systems. It provides several services such as look up, discovery, lease renewal and javaspace service. We can declare the interface J2space which extends Javaspace having primitives which is going to implement as given below. These primitives are implemented here.

Package net.jini.J2space;
Public interface J2Space extends Javaspace {
Boolean delete (Entry tmpl, Transaction txn, long Timeout)
Int countN (Entry tmpl, Transaction txn);
List scan (Entry tmpl, Transaction txn, long Timeout)
List consumingscan (Entry tmpl, Transaction txn, long Timeout)}

4.1.1 CountN ()

This function counts the number of entries which match with specified entry template in JavaSpace server space. This function iterate over all entries in JavaSpace server.

```
While true ()

{
   entry e = iter.next ();
   If (e= = null}
     {
          Break;
      }
Class entryClass = e.getClass
String entryClassName = entryClass.GetName ();
If {
entryClassName.equals((tmpl.getclass()).getname())
{
 totalEntries++
 }
} iter.close
} return totalEntries
```

4.1.2 Delete ()

This function deletes the specified entry from space. This function uses basic method takeIfExists (). In JavaSpace method takeIfExists () takes specified entry from space means read and remove from space.

```
if (myEntry != null)
  {
        return true;
   }
else
  {
       return false;
   }
```

4.1.3 Scan ()

This function returns list of all entries that matches with specified entry template. This function iterate over all entries in space. To iterate over all the entries in JavaSpace server we need to have admin control. So create an object of AdminProxy as

AdminProxy adminProxy = new AdminProxy (theStub, theUuid);

AdminProxy handles all admin related function for JavaSpace server. The arguments of AdminProxy are theStub and theUuid, the theStub is nothing but server and the theUuid is 128 bit universally unique identifier.

```
While true()
{
  entry e = iter.next();
    if (e= = null}
      {
           Break;
       }
Class entryClass = e.getClass
String entryClassName = entryClass.GetName();
If( entryClassName.equals((tmpl.getclass()).getname())
{
 Scanentries.add (e)
 } count++;
}iter.close
}return Scanentries
scanEntries.add (e);
```

4.1.4 ConsumingScan ()

This function takes list of all entries that matches with specified Entry template. This function uses basic method take () and countN () method. In JavaSpace method take () takes specified entry from space means read and remove from space.

noEntries = countN (tmpl, txn);

In Javaspace we can specify the transaction in which entry is involved. So primitives specify entry, involved in transaction txn, Admin will not search overall space. It will search only specified transaction. It reduces the time taken to search the entries. Following code shows the steps involved in implementation of consumingscan () primitive.

```
Entry tempentry;
Int noEntries = CountN (templ, txn)
for (int  i = 0; i < noEntries; i++)
{ tempEntry = take (tmpl, null, timeout);
        scanEntries.add (tempEntry);
    } return scanEntries;
```

5. Notice-board: J2Space Application

J2Space is having all primitives. This application uses primitive of Javaspace and newly implemented primitives. Notice-board application provides functionalities send notice, read notice and gives interactive notifications when notice send by one user to another user. The notice sent is stored in the shared space from which any authorized user can read it. When one user is sending notice to another user it is not necessary that receiver should alive or in login state. When receiver does login it will get the notification that some notices are for him through system tray and notice toaster. This application is implemented in different modules.

These modules are Login, System Tray, Show Notice, Send Notice and Notifying.

Login: This module consist user authentication of application. In this user interface consist of textboxes for Login id and password. Important thing in this is user types two types of user one- User who have authorization to send notice (user type=1) and second- User who have not authorization to send notice (user type=2). After login successful it calls Notifier.java (system tray).

System Tray: This module initially counts and scans the notices using countN () and scan () function. Then show the notification using Notice Toaster, Which have interactive popup window shows abstract of notice. Then it loads the system tray and pop menu, menu My Notice, Send Notice and Exit and its event Listeners.

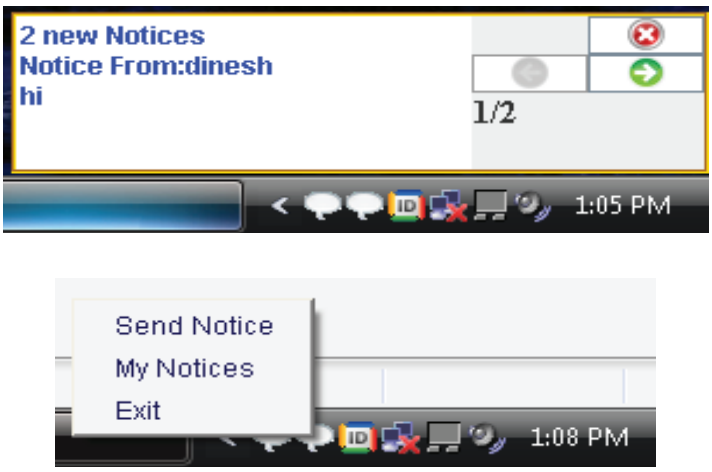

Fig. 4 System Tray and Toaster

J2Space and Javaspace functions used:

CountN (): In this Application primitive used of J2Space is countN () for counting number of specific entries in space. It is not possible to count entries using only JavaSpace functions because in read () function it reads any random specific entry. If we use counter and read, problem will be there, because it may read same entry again.

Read (): Used to read specific entry from space if countN =1
Scan():scan function used to read all entries at same time and display to notify user. If we use other function like read () so according to definition it may return same entry again and if we use take () then it take entry from space and in case user not able to read notice then problem occurs entry deleted from space before read.
Show Notice: This module shows notices in details. It shows single notice at time and next and previous button navigations to view another notices. Initially it scans all user specific notices from space and displays. This module deletes notices which are viewed by user for this purpose a set variable is used. Finally when Close button clicked then all notices which are viewed are deleted from space. Following primitives are used, scan (), delete (), countN ().

Send Notice: This module consist user interface for notice send (i.e. texts for subject, message body and selection of recipient) and Send button on which action event writes (write ()) notice in space. For selection of recipients user ids are load from database. In this module Write () primitive is used.

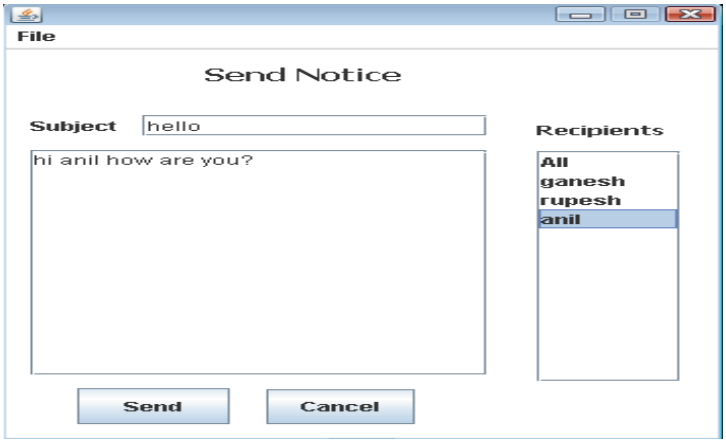

Fig. 5 Send notice window

Notifying: This module consist a space registration, Event Listener and notify to user. In this space is register for Events and Event Listener listens all events on space if write event occur it notifies the user about new notification using Toaster popup message.
Notify () primitive is used for notification

V. CONCLUSION

This paper discusses about various coordination models and coordination languages. Linda is the basic coordination language which uses the shared data space for communication and coordination. It is having some limitations which are recovered in Javaspace. Javaspace is the coordination language which used Java language with coordination primitives. It uses shard space in which java object are stored in serialized form. Jini provides different types of services. Javaspace service is provided by Jini. Javaspace and TSpace is compared and came to know that Javaspace is having very limited primitives. Those primitives are delete (), scan (), consumingscan (), and CountN (). Javaspace is extended as J2Space which contains all primitives of Jasvaspace and newly implemented. For testing of primitives working, Notice-board application is developed in which all primitives are used.

REFERENCES

[1] G.A. Papadopoulos, F. Arbab coordination Models and Languages, CWI, 1998.
[2] S. Ahuja, N. Carriero and D. Gelernter, Linda and Friends, IEEE Computers 19(8), 1986
[3] J-M. Andreoli, H. Gallaire and R. Pareschi, Rule based object coordination
[4] J-M Andreoli, C. Hankin and D. Le Mtayer, Coordination Programming: Mechanism, Models and Semantics, Word scientific, 1996