

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

AUTOMATED GENERATION OF WRAPPERS FOR INTEROPERABILITY

By

Cheng Heng Ngom

March 2000

Thesis Advisor:

Valdis Berzins

Second Reader:

Swapan Bhattacharya

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 1

20000525 050

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
March 2000

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE

Automated Generation Of Wrappers For Interoperability

5. FUNDING NUMBERS

6. AUTHOR(S)

Cheng Heng Ngom

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING
ORGANIZATION REPORT
NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING /
MONITORING
AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

ABSTRACT (maximum 200 words)

Interoperability between software systems is the ability to exchange services from one system to another. In order to exchange services, data and commands are relayed from the service providers to the requesters. Presently, there are some interoperability techniques that aid the exchange of information, ranging from low-level sockets and messaging techniques to more sophisticated middleware technology like object resource brokers. Middleware technology uses higher abstraction than messaging, and can simplify the construction of interoperable applications. It provides a bridge between the service provider and requester by providing standardized mechanisms that handle communication, data exchange and type marshalling. However, in current interoperability techniques, data and services are tightly coupled to a particular server. Furthermore, most developers are trained in developing stand-alone applications rather than distributed applications. This thesis aims at developing a generic interface wrapper that can be used to separate the data and services from the server, and allows the developers to treat distributed data and services as those they are local within an application process space. In addition, the research developed a program to fully automate the process of generating the interface wrapper directly from a specification language such as Prototype System Description Language (PSDL).

14. SUBJECT TERMS

Interoperability with JavaSpace, Jini, Loosely-Coupled Distributed Systems, Prototype System Description Language, Computer Aided Prototype System.

15. NUMBER OF
PAGES
154

16. PRICE CODE

17. SECURITY CLASSIFICATION OF
REPORT

Unclassified

18. SECURITY CLASSIFICATION OF
THIS PAGE

Unclassified

19. SECURITY CLASSIFI- CATION
OF ABSTRACT

Unclassified

20. LIMITATION
OF ABSTRACT

UL

THIS PAGE IS INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

AUTOMATED GENERATION OF WRAPPERS FOR INTEROPERABILITY

Cheng Heng Ngom
Ministry of Defense, Singapore
B.Eng. (Hons),
Nanyang Technological University Singapore, 1985

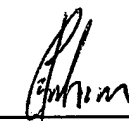
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

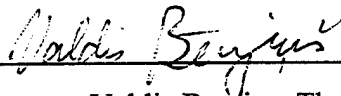
NAVAL POSTGRADUATE SCHOOL
March 2000

Author:

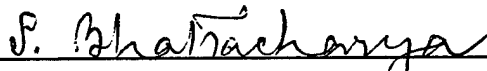


Cheng Heng Ngom

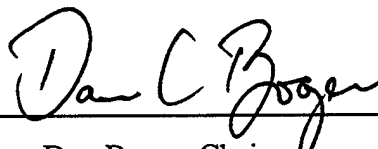
Approved by:



Valdis Berzins, Thesis Advisor



Swapan Bhattacharya, Second Reader



Dan Boger, Chairman
Department of Computer Science

THIS PAGE IS INTENTIONALLY LEFT BLANK

ABSTRACT

Interoperability between software systems is the ability to exchange services from one system to another. In order to exchange services, data and commands are relayed from the service providers to the requesters. Presently, there are some interoperability techniques that aid the exchange of information, ranging from low-level sockets and messaging techniques to more sophisticated middleware technology like object resource brokers (CORBA, DCOM). Middleware technology uses higher abstraction than messaging, and can simplify the construction of interoperable applications. It provides a bridge between the service provider and requester by providing standardized mechanisms that handle communication, data exchange and type marshalling. However, in current interoperability techniques, data and services are tightly coupled to a particular server. Furthermore, most developers are trained in developing stand-alone applications rather than distributed applications. This thesis aims at developing a generic interface wrapper that can be used to separate the data and services from the server, and allows the developers to treat distributed data and services as those they are local within an application process space. In addition, the research developed a program to fully automate the process of generating the interface wrapper directly from a

specification language such as Prototype System Description Language (PSDL).

Table of Contents

I. INTRODUCTION	1
A. BACKGROUND	1
B. CURRENT STATE-OF-THE-ART SOLUTIONS	1
C. MOTIVATION.....	3
D. PROPOSAL.....	3
II. REVIEW OF EXISTING WORKS:	5
A. ORB APPROACHES	5
B. PROTOTYPING	6
C. TRANSACTION HANDLING.....	7
III. THE BASIC MODEL	9
A. JAVASPACE MODEL	9
B. AICG MODEL.....	11
IV. DEVELOPING DISTRIBUTED APPLICATIONS WITH THE AICG TOOL	15
A. DEVELOPMENT PROCEDURES	15
B. INPUT DEFINITION TO THE CODE GENERATOR	17
C. RUNNING THE PSDL2SPACE PROGRAM	23
D. THE SERVER SIDE OF TRACK	24
E. THE CLIENT SIDE OF TRACK WITHOUT EVENT-NOTIFICATION ENABLED.....	26
F. THE CLIENT SIDE OF TRACK WITH EVENT-NOTIFICATION ENABLED	27
V. CHARACTERISTICS OF AICG MODEL	29
A. DISTRIBUTED DATA STRUCTURE AND LOOSELY COUPLED PROGRAMMING	29
B. SYNCHRONIZATION.....	32
C. OBJECT LIFE TIME (LEASES/TIMEOUT).....	34
D. TRANSACTIONS.....	36
1. <i>Jini Transaction Model</i>	36
2. <i>AICG Transaction Model</i>	37
E. AICG EVENT NOTIFICATION.....	38
VI. AICG DESIGN	41
A. AICG ARCHITECTURE	41
B. INTERFACE MODULES	44
1. <i>Entry</i>	44
2. <i>Serialization</i>	47
3. <i>The Actual Distributed Object</i>	48
4. <i>Object Wrapper</i>	49
C. EVENT MODULES.....	51
1. <i>Event Identification Class</i>	51
2. <i>Event Handler</i>	51
3. <i>The Callback Template</i>	53
D. THE TRANSACTION MODULES	53
E. THE EXCEPTION MODULE.....	54
VII. CONCLUSIONS AND RECOMMENDATION	57
A. CONCLUSIONS.....	57
B. RECOMMENDATIONS.....	58

1. <i>Graphical User Interface (GUI)</i>	59
2. <i>Integration With The CAPS System</i>	59
APPENDIX A. PSDL GRAMMAR	61
EXTENSION TO PSDL FOR AICG MODEL	64
APPENDIX B. EXAMPLES OF GENERATED INTERFACE WRAPPER	65
ENTRYAICG.JAVA	65
TRACK.JAVA	66
TRACKENTRY.JAVA	67
TRACKEXT.JAVA.....	68
TRACKEXTCLIENT.JAVA.....	69
TRACKEXTSERVER.JAVA.....	71
NOTIFYAICG.JAVA.....	75
EVENTAICGID.JAVA.....	75
EVENTAICGHANDLER.JAVA	76
TRANSACTIONAICG.JAVA	80
TRANSACTIONMANAGERAICG.JAVA.....	81
EXCEPTIONAICG.JAVA.....	82
SPACEAICG.JAVA	84
POSITION_TYPE.JAVA.....	85
APPENDIX C. PSDL2SPACE CODE LISTINGS	87
AUTOMCODEEXCEPTION.JAVA.....	87
CLASSDEFINITION.JAVA	89
CLASSIDDEFINITION.JAVA	93
DEFINITION.JAVA.....	94
METHODDEFINITION.JAVA	96
READER.JAVA	101
SPACEPROPERTIES.JAVA	103
VARAIBLEDEFINITION.JAVA	106
PSDLDEFINITION.JAVA.....	109
PSDLREADER.JAVA	111
APPBUILDER.JAVA	119
APPENDIX D. SETTING UP THE JAVASPACE ENVIRONMENT	135
EXTRACTED FROM THE NUTS AND BOLTS OF COMPILING AND RUNNING JAVASPACE PROGRAMS BY SUSANNE HUPFER	135
LIST OF REFERENCES	151
INITIAL DISTRIBUTION LIST	153

ACKNOWLEDGEMENTS

My sincerest appreciation and thanks to my thesis advisor, Professor Valdis Berzins. His knowledge of the field is extraordinary and he provided many insightful ideas, assistance, and support.

I would also like to give special thanks to Professor Swapan Bhattacharya for working diligent hours in helping me in the thesis.

- Ngom H. Cheng

THIS PAGE IS INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Interoperability between software systems is the ability to exchange services from one system to another. In order to exchange services, data and commands are relayed from the service providers to the requesters. Current business and military systems are typically 2-tier or 3-tier systems involving clients and servers, each running on different machines in the same or different locations. Current approaches for n-tier systems have no standardization of protocol, data representation, invocation techniques etc. Other problems with interoperability are the implementation of distributed systems and the use of services from heterogeneous operating environments. These include issues concerning sharing of information amongst various operating systems, and the necessity for evolution of standards for using data of various types, sizes and byte ordering, in order to make them suitable for interoperation. These problems make interoperable applications difficult to construct and manage.

B. CURRENT STATE-OF-THE-ART SOLUTIONS

Presently, the solutions attempting to address these interoperability problems range from low-level sockets and messaging techniques to more sophisticated middleware

technology like object resource brokers (CORBA, DCOM). Middleware technology uses higher abstraction than messaging, and can simplify the construction of interoperable applications. It provides a bridge between the service provider and requester by providing standardized mechanisms that handle communication, data exchange and type marshalling. The implementation details of the middleware are generally not important to developers building the systems. Instead, developers are concerned with service interface details. This form of information hiding enhances system maintainability by encapsulating the communication mechanisms from the developers and providing a stable interface services for the developers. However, developers still need to perform significant work in incorporating the middleware's services into their systems. Furthermore, they must have a good knowledge of how to deploy the middleware services to fully exploit the features provided.

Current middleware approaches have another major limitation in the design - the data and services are tightly coupled to the servers. Any attempt to parallelize or distribute a computation across several machines therefore encounters complicated issues due to this tight control of the server process on the data.

C. MOTIVATION

Distributed data structures provide an entirely different paradigm. Here, data is no longer coupled to any particular process. Methods and services that work on the data are also uncoupled from any particular process. Processes can now work on different pieces of data at the same time. So far, building distributed data structures together with their requisite interface has proved to be more daunting than other conventional interoperability middleware techniques. The arrival of JavaSpace has changed the scenario to some extent. It allows easy creation and access of distributed objects. However, issues concerning data getting lost in the network, duplicated data items, out-dated data, external exception handling and handshaking of communication between the data owner and data users are still open. The developers have to devise ways to solve those problems and standardize them between applications.

D. PROPOSAL

The situation concerning interoperability would greatly improve if a developer working on some particular application were provided with the features capable of treating distributed objects as local objects within the application. The developers could then modify the distributed object as if it is local within the process. The changes may, however, still need to be reflected on other

applications using that distributed object without creating any problems related to inconsistency. The present thesis aims at attaining this objective by creating a model of an interface wrapper that can be used for a variety of distributed objects. In addition, by automating the process of generating the interface wrapper directly from the interface specification of the requirement, developers' productivity is greatly improved.

The tool, named the Automated Interface Codes Generator (AICG), has been developed to generate the interface wrapper codes for interoperability, from a specification language called the Prototype System Description Language (PSDL) [LUQ88]. The tool uses the principles of distributed data structure and JavaSpace Technology to encapsulate transaction control, synchronization, and notification together with lifetime control to provide an environment that treats distributed objects as if there were local within the concerned applications.

II. REVIEW OF EXISTING WORKS:

A. ORB APPROACHES

A basic idea for enhancing interoperability is to make the network transparent to the application developers. The existing approaches [BER99] include 1) Building blocks for interoperability, 2) Architectures for unified, systematic interoperability and 3) Packaging for encapsulating interoperability services. These approaches have been assessed using the Kiviat graphs by Berzins [BER99] with various weight factors. The Kiviat graphs give a good summary of the strong and weak points of various approaches. ORB and Jini are currently the more promising technologies for interoperability.

There are however, some concerns with the ORB models. Sullivan [SUL99] provides a more in-depth analysis of the DCOM model, highlighting the architecture conflicts between Dynamic Interface Negotiation (how a process queries a COM services and interface) and Aggregation (component composition mechanism). The interface negotiation does not function properly within the aggregated boundaries. This problem arises out of the sharing of an interface by the components. An interface is shared if the constructor or QueryInterface functions of several components can return a pointer to it. QueryInterface rules state that a holder of a

shared interface should be able to obtain interfaces of all types appearing on both the inner and outer components. However, an aggregator can refuse to provide interfaces of some types appearing on an inner component by hiding the inner component. Thus, QueryInterface fails to work properly with respect to delegation to the inner interface.

Hence, for the ORB approaches, detailed understanding of the techniques is required to design a truly reliable interoperable system. Programmers however, are trained mostly on standalone programming techniques. Addition of specialized network programming models increases the learning as well as development time, with occasional slippage of target deadlines. Furthermore, bugs in the distributed programs are harder to detect and consequences of failure are more catastrophic. An abnormal program may cause other programs to go astray in a connected distributed environment [LUQ88], [LUQ98].

B. PROTOTYPING

The demand for large, high quality systems has increased to the point where a quantum change in software technology is needed [LUQ88]. Rapid prototyping is one of the most promising solutions to this problem. Completely automated generation of prototype from a very high-level language is feasible and in-fact generation of skeleton programming structures is very common in the computer world.

One major advantage of the automatic generation of codes is that it frees the developers from the implementation details by executing specification via reusable components [LUQ88].

In this perspective, an integrated software development environment, named Computer Aided Prototyping System (CAPS) has been developed at the Naval Postgraduate School, for rapid prototyping of hard real-time embedded software systems, such as missile guidance systems, space shuttle avionics systems, and military Command, Control, Communication and Intelligence (C3I) systems [LUQ92]. Rapid prototyping uses rapidly constructed prototypes to help both the developers and their customers visualize the proposed system and assess its properties in an iterative process. The heart of CAPS is the Prototype System Description Language (PSDL). It serves as an executable prototyping language at the specification or design level and has special features for real-time system design. Building on the success of computer aided rapid prototyping system (CAPS) [LUQ92], the AICG model also uses the PSDL for the specification and automates the generation of interface codes with the objective of making the network transparent from the developer's point of view.

C. TRANSACTION HANDLING

Building a networked application is quite different from building a stand-alone system in the sense that many

additional issues need to be taken care of for smooth functioning of a networked application. The networked systems are also susceptible to partial failures of computation, which can leave the system in an inconsistent state.

Proper transaction handling is essential to control and maintain concurrency and consistency within the system. Yang [YAN99], examined the limitation of hard-wiring concurrency control (CC) into either the client or the server. He found that the scalability and flexibility of these configurations is greatly limited. Hence, he presented a middleware approach: an external transaction server, which carries out the concurrency control policies in the process of obtaining the data. Advantages of this approach are 1) transaction server can be easily tailored to apply the desired CC policies of specific client applications. 2) The approach does not require any changes to the servers or clients in order to support the standard transaction model. 3) Coordination among the clients that share data but have different CC policies is possible if all of the clients use the same transaction server.

The AICG model uses the same approach, by deploying an external transaction manager provided by SUN in the JINI model [JOY99]. All transactions used by the clients and servers are created and overseen by the manager.

III. THE BASIC MODEL

A. JAVASPACE MODEL

JavaSpace model is a high-level coordination tool for gluing processes together in a distributed environment. It departs from conventional distribution techniques using message passing between processes or invoking methods on remote objects. The technology provides a fundamentally different programming model that views an application as a collection of processes cooperating via the flow of freshly copied objects into and out of one or more spaces. This space-based model of distributed computing has its roots in the Linda coordination language [GEL85] developed by Dr. David Gelernter at Yale University.

A space is a shared, network-accessible repository for objects. Processes use the repository as a persistent object storage and exchange mechanism. As shown in figure 1, processes perform simple operations to write new objects into space, take objects from space, or read (make a copy of) objects in a space. While taking or reading objects, processes use a simple value-matching lookup to find the objects that matter to them. If a matching object is not found immediately, then a process can wait until one arrives. Unlike conventional object stores, processes do not modify objects in the space or invoke their methods

directly. To modify an object, a process must explicitly remove it, update it, and reinsert it into the space. During the period of updating, other processes requesting for the object will wait until the process writes the object back to the space.

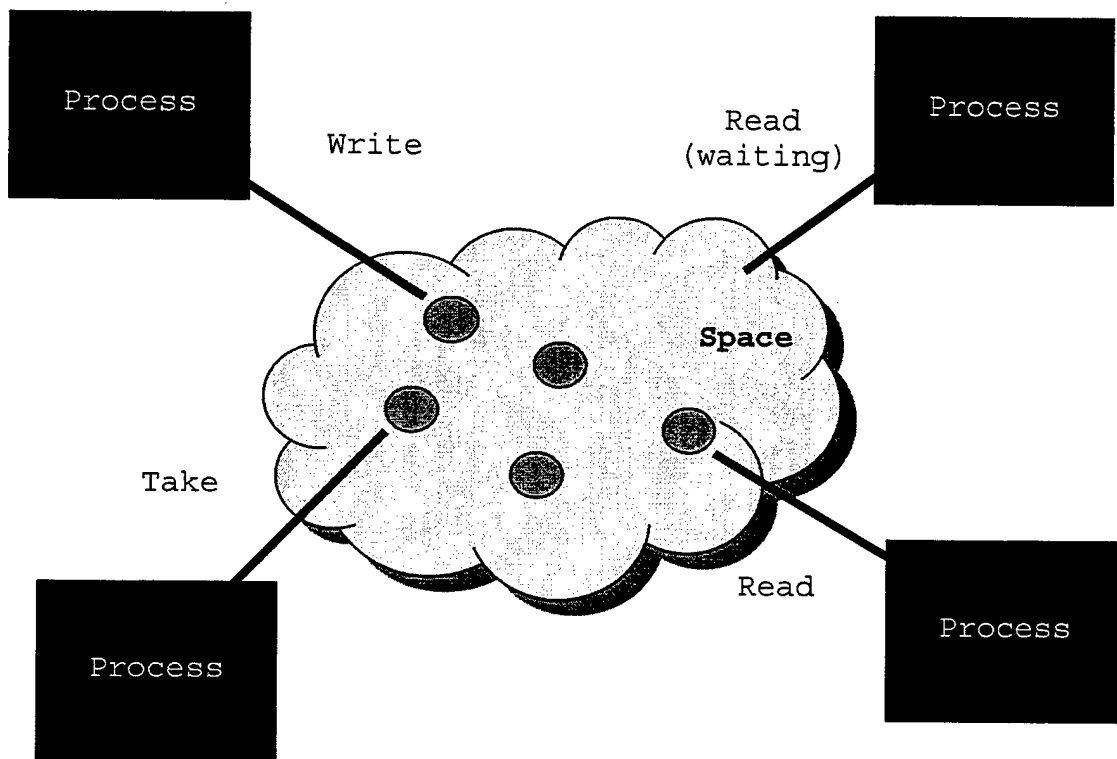


Figure 1, JavaSpace operations

Key Features of JavaSpace:

- Spaces are persistent: Spaces provide reliable storage for objects. Once stored in the space, an object will remain there until a process explicitly removes it.

- Spaces are transactionally secure: The Space technology provides a transaction model that ensures that an operation on a space is atomic. Transactions are supported for single operations on a single space, as well as multiple operations over one or more spaces.
- Spaces allow exchange of executable content: While in the space, objects are just passive data. However, when we read or take an object from a space, a local copy of the object is created. Like any other local object, we can modify its public fields as well as invoke its methods.

B. AICG MODEL

The interoperability approach underlying the AICG model, proposes a tool for building distributed applications. The tool is designed to generate interface wrappers for data structures or objects that need to be shared, and are particularly useful for applications that can model as flows of objects through one or more servers. Build on top of JavaSpace, the AICG model hides the space and its implementation details entirely from the application. The interface wrapper allows applications to treat distributed data structures or objects as local within the application space. This enhances interoperability by

making the network transparent to the application developers.

The interface wrappers are generated from an extension of a prototype description language called Prototype System Description Language (PSDL).

Some of the salient features of the AICG model are:

- Distributed objects are treated as local objects within the application process. The application codes need not depend on how the object is distributed, since the local object copy is always synchronous with the distributed copy. (Chapter V)
- Synchronization with various applications is automatically handled. Since the AICG model is based on the space transaction secure model, all operations are atomic. Deadlock is prevented automatically within the interface by having only a single distributed copy, and through transaction control. (Chapter VI, XI)
- Any type of object can be shared as long as the object is serializable. Any data structure and object can be distributed as long as it obeys and implements the java serializable feature (Chapter VI, Section B-2).
- Every distributed object has a lifetime. The distributed object lifetime is a period of time

guaranteed by the AICG model for storage and distribution of the object. The time can be set by developer. (Chapter V, Section C)

- All write operations are transaction secure by default. AICG transactions are based on the Atomicity, Consistency, Isolation, and Durability (ACID) properties. (Chapter V, Section D)
- Clients can be informed of changes to the distributed object through the AICG event model. A client application can subscribe for change notification, and when the distributed object is modified, a separate thread is spawned to execute the callback method defined by the developer. (Chapter V, Section E)
- The wrapper codes are generated from high-level descriptive languages; hence, they are more manageable and more maintainable.

THIS PAGE IS INTENTIONALLY LEFT BLANK

IV. DEVELOPING DISTRIBUTED APPLICATIONS WITH THE AICG TOOL

This section describes the steps for developing distributed applications using the AICG model. An example of a C4ISR application is introduced in section 4.1.2 to aid the explanation of the process. The same example will be used throughout this paper.

A. DEVELOPMENT PROCEDURES

The developer starts the development process by defining shared objects using the Prototype System Description Language (PSDL). The PSDL is processed through a code generator (PSDLtoSpace) to produce a set of interface wrapper codes (figure 2). The interface wrapper contains the necessary codes for interaction between the applications and the space without the need for the developers to be concerned with the writing and removing of objects in the space. The developers can treat shared or distributed objects as local objects, where synchronization and distribution is automatically handled by the interface codes.

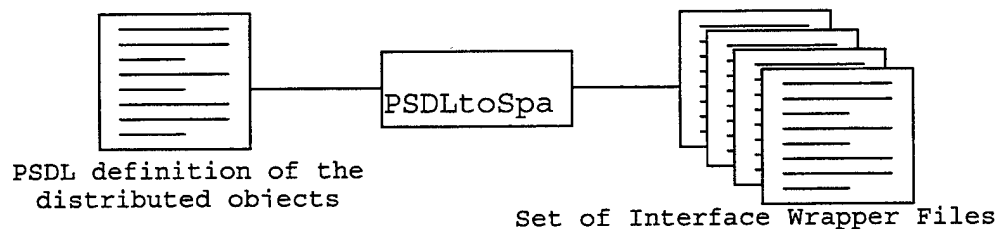


Figure 2, PSDL to Space

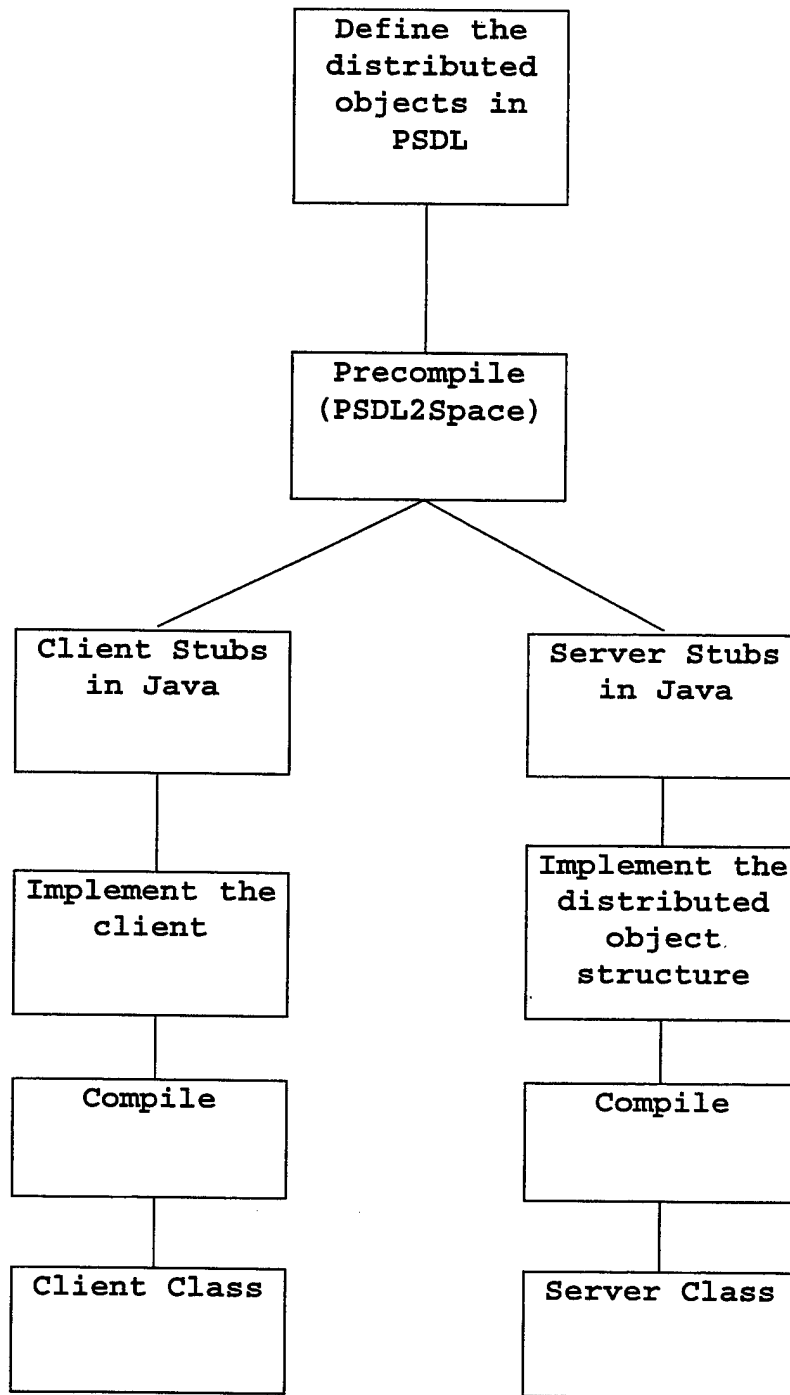


Figure 3, Generating the interface codes

The complete cycle for generating the interface codes is shown in figure 3.

B. INPUT DEFINITION TO THE CODE GENERATOR

The following example demonstrates the development of one of the many distributed objects in the C4ISR system. Airplane positions picked up from the sensors are processed to produce track objects. These objects are distributed over a large network and used by several clients' stations for displaying the positions of planes. Each track or plane is identified by track number. The tracks are 'owned' by a group of track servers, and only the track servers can update the track positions and its attributes. The clients only have read access on the track data. Figure 4 shows the PSDL codes for the track object and its methods. Figure 5 shows the PSDL codes for the **Track_list** object and its methods.

The PSDL grammar used for the AICG is an extended version of the original PSDL grammar (Appendix A). PSDL model is very extensive and can be used to model an entire distributed system. However, the AICG only used a portion of the PSDL to describe the interface between systems. In another word, interactions between applications are defined using the PSDL but not the application itself. Because of this, slight modifications on the PSDL grammar were needed.

The complete listing of the changes in the grammar statements can also be found in Appendix A.

The track PSDL starts with the definition of a type called **track**. It has only one identification field **tracknumber**. Of course, the **track** objects can have more than one field, but only one field is in this case is used to uniquely identify any particular **track** object. The type **track_list** shown in figure 5, on the other hand, does not need an identification field since there is only one **track_list** object in the whole system. **Track_list** is used to keep a list of all the active tracks **tracknumber** in the system at that moment in time.

All the operators (methods) of the type are defined immediately after the specification. Each method has a list of *input* and *output* parameters that define the arguments of the method. The most important portion in the method declaration is the *implementation*. The developer must be able to define the type of operation the method is supposed to perform. The operations are *constructor* (used to initialize the class), *read* (no modification to any field in the class) and *write* (modification is done to one or more fields in the class). These are necessary, as the code generated will encapsulate the synchronization of the distributed objects.

<pre> Type track SPECIFICATION tracknumber: integer END OPERATOR track SPECIFICATION INPUT x:integer END IMPLEMENTATION SPACE PROPERTY SPACEMODE= CONSTRUCTOR END END OPERATOR getID SPECIFICATION OUTPUT x:integer END IMPLEMENTATION SPACE PROPERTY SPACEMODE= READ END END OPERATOR setCallsign SPECIFICATION INPUT sign: string END IMPLEMENTATION SPACE PROPERTY SPACEMODE= WRITE PROPERTY TRANSACTIONTIME = 300 END END OPERATOR getCallsign SPECIFICATION OUTPUT sign: string END IMPLEMENTATION SPACE PROPERTY SPACEMODE= READ END END </pre>	<pre> OPERATOR setPosition SPECIFICATION INPUT post : position_type END IMPLEMENTATION SPACE PROPERTY SPACEMODE = WRITE PROPERTY TRANSACTIONTIME = 2000 END END OPERATOR getPosition SPECIFICATION OUTPUT post : position_type END IMPLEMENTATION SPACE PROPERTY SPACEMODE = READ END END IMPLEMENTATION SPACE PROPERTY SPACENAME= DODSpaces PROPERTY OWNERSHIP = YES PROPERTY SECURITY = SERVER PROPERTY LEASE = 12000 PROPERTY CLONE = MANY PROPERTY NOTIFY = NO PROPERTY RETRY = 10 END </pre>
---	--

Figure 4, Track example in PSDL

<pre> TYPE track_list SPECIFICATIO END OPERATOR track_list SPECIFICATION END IMPLEMENTATION SPACE PROPERTY SPACEMODE= CONSTRUCTOR END END OPERATOR getID SPECIFICATION INPUT index: integer OUTPUT x:integer END IMPLEMENTATION SPACE PROPERTY SPACEMODE= READ END END OPERATOR setNewID SPECIFICATION INPUT id: integer END IMPLEMENTATION SPACE PROPERTY SPACEMODE= WRITE END END OPERATOR removeID SPECIFICATION INPUT id: integer END IMPLEMENTATION SPACE PROPERTY SPACEMODE= WRITE PROPERTY TRANSACTIONTIME = 2000 END END END </pre>	<pre> IMPLEMENTATION SPACE PROPERTY SPACENAME= DODSpaces PROPERTY OWNERSHIP = YES PROPERTY SECURITY = SERVER PROPERTY LEASE = 0 PROPERTY CLONE = ONE PROPERTY NOTIFY = YES PROPERTY RETRY = 5 END TYPE position_type SPECIFICATION END IMPLEMENTATION JAVA position.java END </pre>
--	--

Figure 5, Track list example PSDL

The other field in the implementation portion of the method, is *transactiontime*. *transactiontime* defines the upper limit in milliseconds within which the operation must be completed. The transaction property is discussed in detail in Chapter VI, Section D. Method Implementation property fields include:

```
SPACEMODE = CONSTRUCTOR/ : Type of operation
                        READ /WRITE
TRANSACTIONTIME = 999    : Upper limit of the operation
                        : in ms.
                        : Transaction is disabled by
                        : setting the value to 0.
```

At the end of each class is the implementation section. This section is the class-wide attributes of the ACIG properties. The explanation of the attributes is:

```
SPACENAME= DODSpaces      : Which Java Space to be used
OWNERSHIP = YES/NO        : Every object has at most one
                        : owner?
SECURITY = SERVER         : Only Server application can
                        : modify the object
                        /Client : All application can modify
                        : the object
```

LEASE = 999	: Object life time in ms. For
	: lease that last forever, set
	: it to 0
CLONE = ONE	: There is only one object of
	: this type
	: in the space at a time
/MANY	: There can be many object of
	: the this type in the space
	: simultaneously.
NOTIFY = YES	: Notification of changes
	: allowed.
/NO	: Notification of changes not
	: allowed.
RETRY = n	: Retry all communication
	: operations on the space for
	: n times, before raising an
	: exception.

C. RUNNING THE PSDL2SPACE PROGRAM

Each distributed class in the PSDL will generate a set of interface wrapper files after executing the PSDL2Space program.

The command line for PSDL2Space is:

- PSDL2Space <destination dir> <data dir> <psdl file>

Figure 6 below shows the files created for executing the command:

```
C:> PSDL2Space interfaceAICG aicgDir track.psd1
```

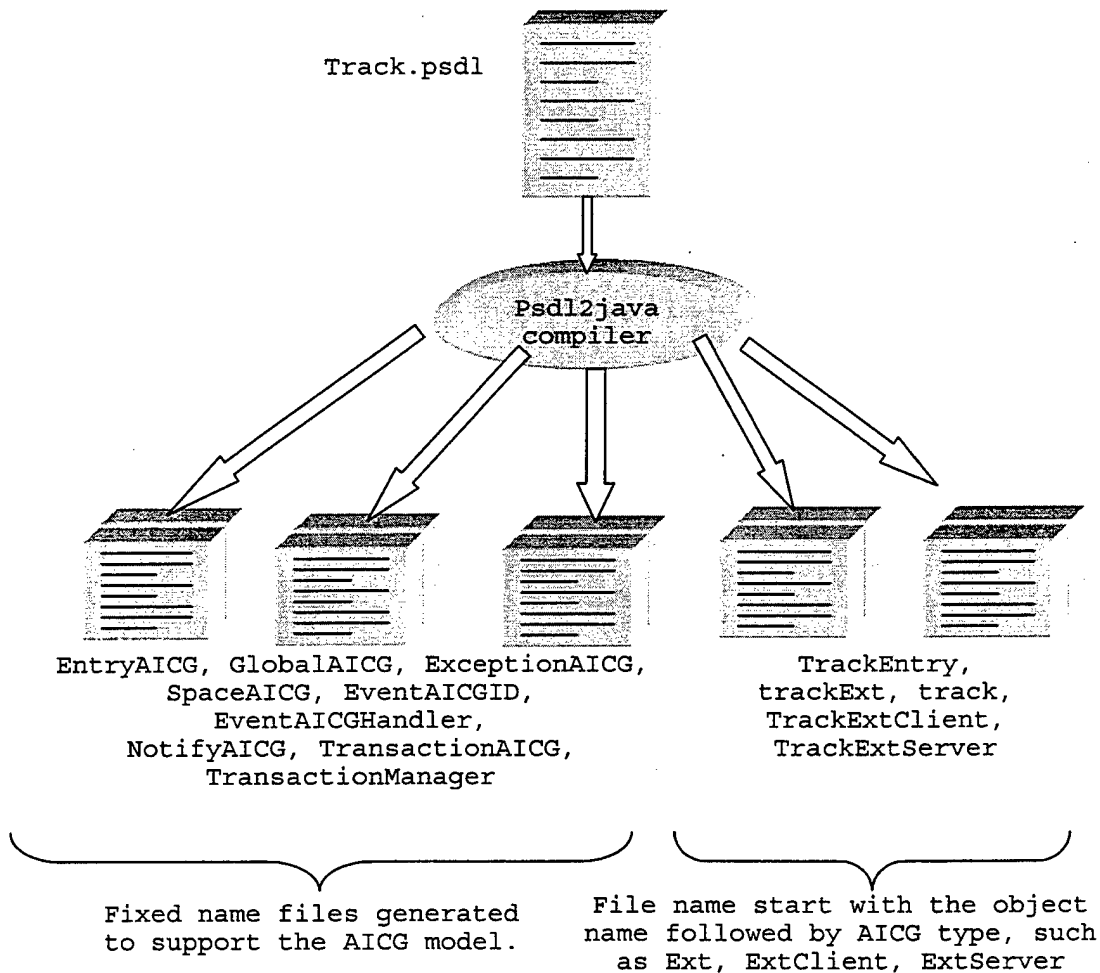


Figure 6, AICG generated

Most of the generated files can be ignored except the following:

- I. `Track.java`: this file contains the skeleton of the fields and the methods of the track class. The user is supposed to fill the body of the methods.
- II. `TrackExtClient.java`: this is the wrapper class that the client initialized and used instead of the track class.
- III. `TrackExtServer.java`: this is the wrapper class that the server initialized and used in replace for the track class.
- IV. `NotifyAICG.java` : this class must be extended or implemented by the application if event-notification and call-back are needed.

The methods found in the `trackExtClient` and `trackExtServer` have the same method names and signatures of the track class. In fact, the track class methods are been called within `trackExtClient` or `trackExtServer`.

D. THE SERVER SIDE OF TRACK

Every AICG server must have a main program that initializes the AICG environment and creates the distributed objects. An example of a small portion of the server program

is given below. The full example listing can be found in Appendix B. It is noted that the once the distributed object is instantiated, it behaves like any local object that be invoked or modified.

```
public class TrackServer implements Runnable
{

    public void run(){
        // TODO: Add initialization code here
        trackExtServer trk[];
        position_type pos;

        try{
            // Start the transaction manager
            trackExt.trackStartTxn();
            // create array of for 10 tracks
            trk = new trackExtServer[10];

            rd = new Random();
            // create 10 tracks with random position
            for (int i=0; i <10; i++){
                // create the tracks
                trk[i] = new trackExtServer(i);
                // generate a random number
                pos = new position_type
                    (rd.nextInt(),rd.nextInt());
                // set the position
                trk[i].setPosition(pos);
            }
        }catch (Exception e){
            System.err.println("Exception");
            e.printStackTrace();
        }
    }
}
```

Figure 6, An example of Track Server

E. THE CLIENT SIDE OF TRACK WITHOUT EVENT-NOTIFICATION ENABLED

The client program example consists of a single Java class, TrackClient. This class provides the main method that performs only one main task; Initialize the trackExtClient object. Of course, the client must know the track number in advance before it can locate the distributed object. Since event-notification is not enabled, the client has to keep polling on the contents of the track. Part of TrackClient is shown below in figure 8. The full listing of TrackClient is found in Appendix B.

```
public class TrackClient implements Runnable
{
    public void run(){
        // initialize and search for the track in the space
        try{
            trk = new trackExtClient[10];
            //initialize the track clients
            for (int i=0; i <10; i++){
                trk[i] = new trackExtClient(i);
            }
            // loop every 6 seconds
            while (true){
                Thread.sleep(6000);
                for (int i =0; i <10; i++){
                    System.out.print("ID :" + trk[i].getID());
                    System.out.println(" X :" +
                        trk[i].getPosition());
                }
            }
        }catch (Exception e){
            System.err.println("Exception");
            e.printStackTrace();
        }
    }
}
```

Figure 8, Example of Track Client without Notification

F. THE CLIENT SIDE OF TRACK WITH EVENT-NOTIFICATION ENABLED

The client program is the same as the one above with the only exception is that the notification is enabled. Hence, whenever the server changes the contents of the track object, listenerAICGEvents method will be invoked.

```
public class TrackClient implements Runnable, notifyAICG
{
    public void run(){
        try{
            // initialize event notification
            handler = new eventAICGHandler(this);
        }catch (exceptionAICG e){
        }
        try{
            trk = new trackExtClient[10];
            // enable this track with notification for changes
            trk[i].setEvent(handler);
        }
    }
    // listenerAICGEvent will be invoked when changes occur
    public void listenerAICGEvents(Object obj){
        try {
            trackExtClient trkClient = (trackExtClient) obj;
            System.out.println("Notified detected");
            System.out.print("ID :" + trkClient.getID());
            System.out.println("Posit" + trkClient.
                getPosition());
        }catch (Exception e){
            System.err.println("Exception");
        }
    }
}
```

Figure 9, Example of Track Client with Notification

THIS PAGE INTENTIONALLY LEFT BLANK

V. CHARACTERISTICS OF AICG MODEL

A. DISTRIBUTED DATA STRUCTURE AND LOOSELY COUPLED PROGRAMMING

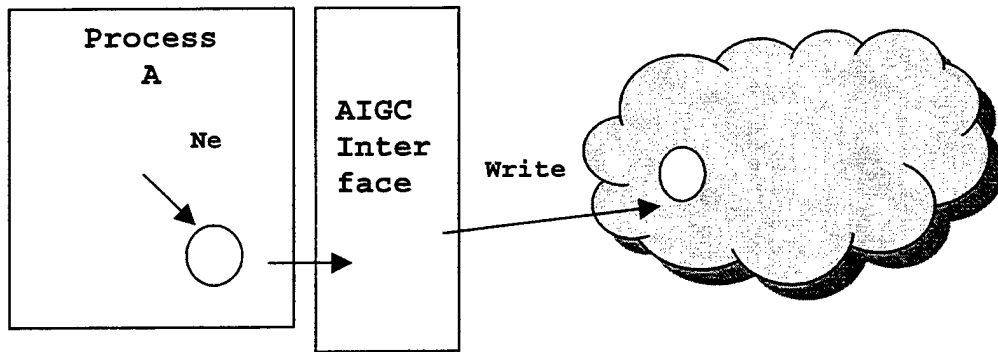
Conceptually a distributed data structure is one that can be accessed and manipulated by multiple processes at the same time without any consideration for any machine physically executing those processes. In most distributed computing models, distributed data structures are hard to achieve. Message passing and remote method invocation systems provide a good example of the difficulty. Most of the systems tend to keep data structure behind one central manager process, and processes that want to perform work on the data structure must "wait in line" to ask the manager process to access or alter a piece of data on their behalf. Attempts to parallelize or distribute a computation across more than one machine face bottlenecks since data are tightly coupled by the one manager process. True concurrent access is rarely achievable.

Distributed data structures provide an entirely different approach where we uncouple the data from any particular process. Instead of hiding data structure behind a manager process, we represent data structures as collections of objects that can be independently and

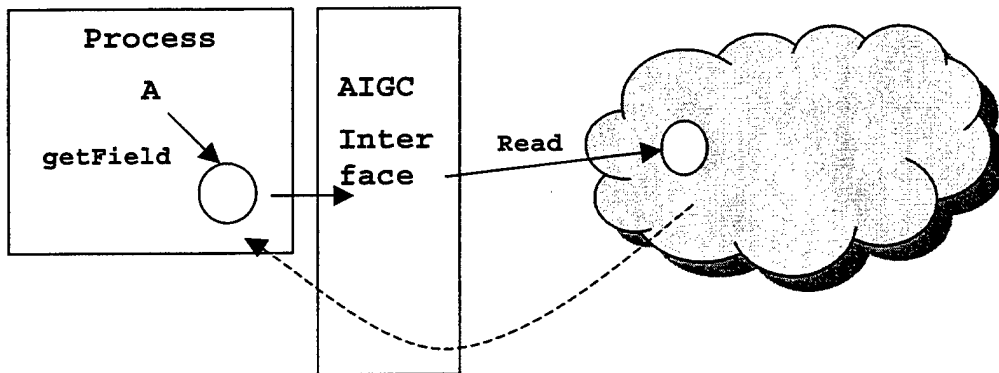
concurrently accessed and altered by remote processes. Distributed data structures allow processes to work on the data without having to wait in line if there are no serialization issues.

The distributed protocol for modification ensures synchronization by enforcing that a process wishing to modify the object has to physically remove it from the space, alter it and write it back to the space. There can be no way for more than one process to modify an object at the same time. However, this does not prevent other processes from overwriting the corrected data. For example, in the normal JavaSpace, process A instead of performing a "take" follow by a "write operation, the programmer wrote a "read" operation, followed by a "write" operation. This results in 2 copies of the object in the Space. The AICG model prevents this by encapsulating the 3 basic commands from the developers. All modification on the object are automatically translated to "take", followed by "write" and all operations that access the fields of the distributed object are translated to "read". These ensure that local data are up-to-date and serialization is maintained (figure 10).

Creating a Distributed Object



Reading a field of a distributed object



Updating a distributed object

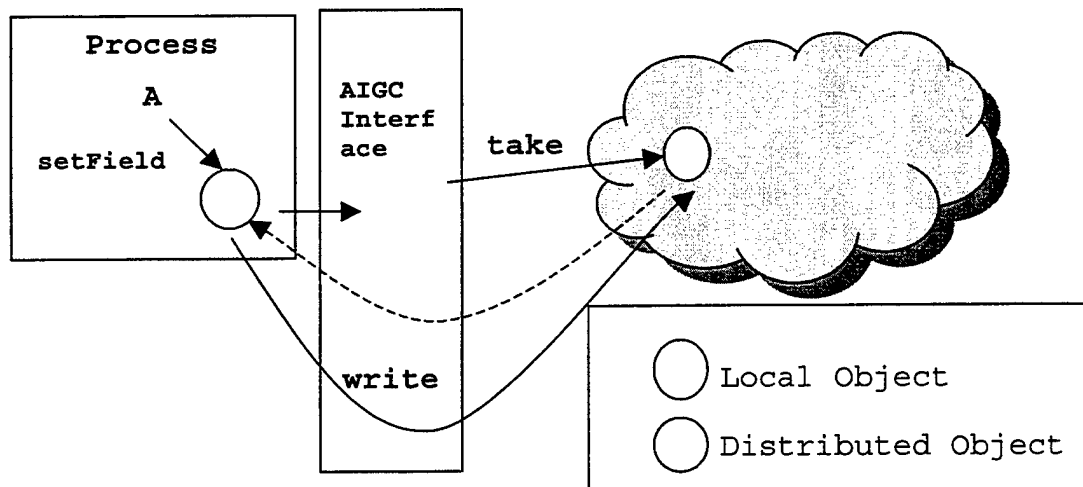


Figure 10, Interaction between Space through AIGC Interface

Loosely-coupled programming also has its own limitations. Distributed objects may be lost if a process removes it from the space and subsequently crashes or is cut off from the network. Similarly, the system may enter a deadlock state if processes request more than one distributed object while, at the same time, holding on to distributed objects required by other processes. In cases like this, the AICG model groups multiple operations into a transaction to ensure that either all operations are completed or none occurs, thereby maintaining the integrity of the application. Hence, deadlock is prevented through transaction control. The application can retry the operation immediately or wait for a random time before performing the operation again.

B. SYNCHRONIZATION

Synchronization plays a crucial role in any design of distributed application. Inevitably, processes in a distributed system need to coordinate with one another and avoid bringing the system into an unstable state such as deadlock. Creating distributed applications with AICG can significantly ease the burden of process synchronization since synchronization is already built into the AICG operations. Multiple processes can read an object in a space

at any time, but when a process wants to update an object, it has to remove it from the space and thereby gain exclusive access to it first. Hence, coordinated access to objects is enforced by the AICG interface doing *read*, *take* and *write* operations.

More advanced and complex synchronization schemes can be easily build upon from the basic atomic features of the AIGC operations. An example is semaphores. Semaphores, a synchronization construct that was first used to solve concurrency problems in operating systems, are commonly found in multithreaded programming languages, but are more difficult to achieve in distributed systems. Semaphores are typically implemented as integer counters that require special language or hardware support to ensure the atomic properties of the *UP* (signal) and *DOWN* (wait) operations. Using AIGC space model, we could easily implement a semaphore as a shared variable that holds an integer counter. By assigning a distributed variable or object as a semaphore, groups of distributed objects can be synchronized. Hence, the AIGC model permits the developers to develop more complicated distributed applications without being concerned about synchronization and deadlock. Furthermore, all operations within the AIGC model can impose transaction control with timeout monitoring. After the

timeout period, the transaction would rollback the application to a stable state.

C. OBJECT LIFE TIME (LEASES/TIMEOUT)

Leasing provides a methodology for controlling the life span of the distributed objects in the AICG space. This allows resources to be freed after a fixed period. This model is beneficial in the distributed environment, where partial failure can cause holders of resources to fail thereby disconnecting them from the resources before they can explicitly free them. In the absence of a leasing model, resources could grow without bound.

There are other constructive ways to harness the benefit of the leasing model besides using it as a garbage collector. As for example, in a real-time system, the value of the information regarding some distributed objects becomes useless after certain deadlines. Accessing obsolete information can be more damaging in this case. By setting the lease on the distributed object, the AICG model automatically removes the object once the lease expires or the deadline is reached.

JavaSpaces allocate resources that are tied to leases. When a distributed object is written into a space, it is granted a lease that specifies a period for which the space

guarantees its storage. The holder of the lease may renew or cancel the lease before it expires. If the leaseholder does neither, the lease simply expires, and the space removes the entry from its store.

The AICG model simplified the JavaSpace lease model into two configurations. These are:

- I. Generally, the distributed object lasts forever as long as the space exists, even if the leaseholder (the process that creates the object) has died. This configuration is enabled by setting the *SPACE* lease property in the Implementation to 0.
- II. In the real-time environment, the distributed object lasts for a fixed duration of *x* ms specified by the object designer. To keep the object alive, a write operation must be performed on the object before the lease expires. This configuration is set through the *SPACE* lease property in the Implementation to the time in ms required.

Hence, the developer must provide due consideration towards leasing while developing the application. If an object has a lifetime, it must be renewed before it expires. In the AICG model, renewal is done by calling any method

that modifies the object. If no modification is required, the developer can consider defining a dummy method with the spacemode set to "write". Invoking that method will automatically renew the lease.

D. TRANSACTIONS

The AICG model uses the Jini Transaction model, which provides generic services concerning transaction processing in distributed computing environment.

1. Jini Transaction Model

All transactions are overseen by a transaction manager. When a distributed application needs operations to occur in a transaction secure manner, the process asks the transaction manager to create a transaction. Once a transaction has been created, one or more processes can perform operations under the transaction. A transaction can complete in two ways. If a transaction commits successfully, then all operations performed under it are complete. However, if problems arise, then the transaction is aborted and none of the operations occurs. These semantics are provided by a two-phase commit protocol that is performed by the transaction manager as it interacts with the transaction participants.

2. AICG Transaction Model

AICG model encapsulates and manages the transaction procedures. All operations on the distributed object can be either with transaction control or without. Transaction control operations are controlled with a default lease of 2 seconds. This default value of leasing time may, however, be changed by the user. The transaction lease is kept by the transaction manager as part of the leased resource, and when the lease expires before the operation being committed, the transaction manager aborts the transaction.

The AICG model by default, enables all transaction for write operations and the transaction lease time is two seconds. The developer can modify the lease time through the PSDL SPACE *transactiontime* property.

```
PROPERTY transactiontime = 0 : Disable transaction for  
                                that method  
/n : Set the lease time to n  
                                milliseconds.
```

No read operations in the AICG model have the transaction enable. However, the user can enable it by using the property *transactiontime* with the upper limit in transaction time for the read operation.

E. AICG EVENT NOTIFICATION

In the distributed and loosely-coupled programming environment, it is desirable for an application to react to changes or arrival of newly distributed objects instead of "busy waiting" for it through polling. AICG provides this feature by introducing a callback mechanism that invokes user-defined methods when certain conditions are met.

Java provides a simple but powerful event model based on event sources, event listeners and event objects. An event source is any object that "fires" an event, usually based on some internal state change in the object. In this case, writing an object into space would generate an event. An event listener is an object that listens for events fired by an event source. Typically, an event source provides a method whereby listeners can request to be added to a list of listeners. Whenever an event source fires an event, it notifies each of its registered listeners by calling a method on the listener object and passing it an event object.

Within a Java Virtual machine (JVM), an application is guaranteed not to miss an event fired from within. Distributed events on the other hand, had to travel either, from one JVM to another JVM within a machine or between machines networked together. Events traveling from one JVM

to another may be lost in transit, or may never reach their event listener. Likewise, an event may reach its listener more than once.

Space-based distributed events are built on top of the Jini Distributed Event model, and the AICG event model further extends it. When using the AICG event model, the space is an event source that fires events when entries are written into the space matching a certain template an application is interested in. When the event fires, the space sends a remote event object to the listener. The event listener codes are found in one of the generated AICG interface wrapper files. Upon receiving an event, the listener would spawn a new thread to process the event and invoke the application callback method. This allows the application codes to be executed without involving the developer in the process of event-management.

There are a few steps for setting up AICG event for a particular application. Firstly, the distributed objects must have the SPACE properties for *Notification* set to yes. One of the application classes must *implement* (java term for inherit) the *notifyAICG* abstract class. The *notifyAICG* class has only one method, which is the callback method. The user class must override this method with the codes that need to be executed when an event fires. More details on the

implementation of the notifyAICGHandler have been included in the next chapter.

VI. AICG DESIGN

This section explains the design of the AICG and the codes that are generated from `psdl2java` program. The interface wrapper codes examples used in this section are generated from the track PSDL of figure 4.

A. AICG ARCHITECTURE

The AICG architecture consists of four main modules: Interface modules, Event modules, Transaction modules and the Exception module. The interface modules implement the distributed object methods and communicate directly with the application. In reference to the example, the interface modules for the `track.psd1` are `entryAICG`, `track`, `trackExt`, `trackExtClient`, `trackExtServer`. Instead of creating the actual object (`track`), the application should instantiate the interface class either the `trackExtClient` or `trackExtServer`. Event modules (`eventAICGID`, `evenAICGHandler`, `notifyAICG`) handle external events generated from the JavaSpace that are of interest to the application. Transaction modules (`transactionAICG`, `transactionManagerAICG`) support the interface module with transaction services. Lastly, the exception module (`exceptionAICG`) defines the possible types of exceptions

that can be raised and need to be handled by the application. Figure 11 shows the architecture of the generated interface wrapper and the interaction with the other modules and the application.

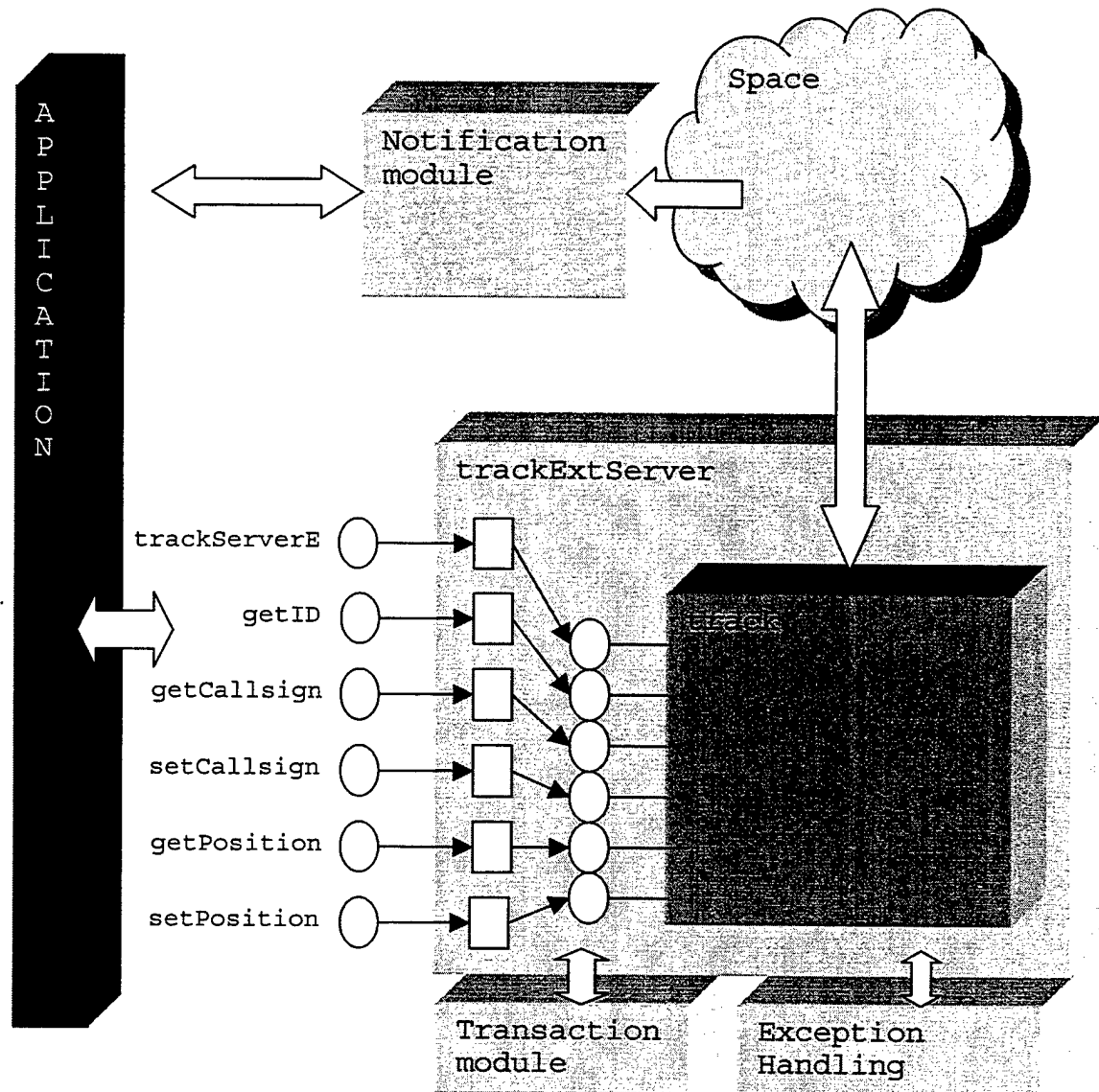


Figure 11, Architecture of the generated interface wrapper and the interaction with the other modules and application

Each time the application instantiate a track class by creating a new trackExtServer, the following events take place in the Interface:

1. An Entry object is created together with the track object by the trackExtServer. The tack object is placed into the Entry object and stored in the space.
2. Transaction Manager is enabled.
3. The reference pointer to trackExtServer is returned to the application.

Each time a method (getID, getCallsign, getPosition) that does not modify the contents of the object is invoked, the following events take place in the Interface:

1. When the application invokes the method through the Interface (trackExtServer/trackExtClient).
2. The Interface performs a Space "get" operation to update the local copy.
3. The method is then executed on the updated copy of the object to return the value back to the application.

Each time a method (*setCallsign*, *setPosition*), which does modify the contents of the object is invoked, the following events take place in the Interface:

1. The application invokes the method through the Interface.
2. The interface performs a Space "take" operation, which retrieves the object from the space.
3. The actual object method is then invoked to perform the modification.
4. Upon completion of the modification, the object is returned to the space by the interface using a "write" operation.

B. INTERFACE MODULES

The interface modules consist of the following modules; an entry (*entryAICG*) that are stored in space, the actual object (*track*) that are shared and the object wrapper (*trackExt*, *trackExtClient*, *trackExtServe*).

1. Entry

A space stores *entries*. An entry is a collection of typed objects that implements the Entry interface. The base class of the AICG distributed object is shown in figure 11.

```

public abstract class entryAICG implements Entry
{
    // main identification number
    public Integer entryID;

    // required by JavaSpace default constructor
    public entryAICG( ) {
    }

    public entryAICG(int id){
        entryID = new Integer(id);
    }

    // return the object stored in the entry
    public abstract Object getObject( );
}

```

Figure 12, An example of entryAICG class

The Entry interface is empty; it has no methods that have to be implemented. Empty interfaces are often referred to as "marker" interfaces because they are used to mark a class as suitable for some role. That is exactly what the Entry interface is used for, to mark a class appropriate for use within a space.

All entries in the AICG extend from this base class. It has one main public attribute, an identifier and an abstract method that returns the object. Any type of object can be stored in the entry. The only limitation is that the object must be serializable. Serializability allows the java virtual machine to pass the entire object by value instead of by reference. Here is an example "track" entry codes generated by the AICG from the PSDL file in figure 4. The

interface contains the object track in one of the field and an ID.

```
public abstract class trackEntry extends entryAICG
{
    // ID is required if there are more than one similar object in
    // the space
    public Integer aicgID1;

    // track object
    public track data;

    // default Constructor
    public trackEntry(){ }

    // Constructor with information extracted from the track PSDL
    // file.
    public trackEntry(int aid, Integer aicgID1, track inData){
        super(aid);
        data = inData;
        this.aicgID1 = aicgID1;
    }
    public Object getObject(){
        return data;
    }
}
```

Figure 13, An example of Entry class

All Entry attributes are declared as publicly accessible. Although it is not typical of fields to be defined in public in object-oriented programming style, the associative lookup is the way the space-based programs locate entries in the space. To locate an object in space, a template is specified that matches the contents of the fields. By declaring entry fields public, it allows the space to compare and locate the object. AICG encourage

object-oriented programming style by encapsulating the actual data object into the entry. The object attributes can then be declared as private and made accessible only through clearly defined public methods of the object.

2. Serialization

Each distributed interface object is a local object that acts as a proxy to the remote space object. It is not a reference to a remote object but instead a connection passes all operations and value through the proxy to the remote space. All the objects must be serializable in order to meet this objective. The Serializable interface is "marker" interface that contains no methods and serves only to mark a class as appropriate for serialization. The Serializable interface is shown in figure 14 together with the example of track class implementing the interface.

```
public abstract interface Serializable {
    // this interface is empty
}

// An example of the track class implementing the interface
//Serializable

public class track implements Serializable {
    // since Serializable is a marker interface
    // no methods need to be override.
}
```

Figure 14, Serializable interface class

3. The Actual Distributed Object

We now look at the actual objects that are shared between the servers and clients. The PSDL2Space generates a skeleton version of the actual class with the methods names and its arguments. The body of the methods and its fields need to be filled by the developers. The track class generated is shown in figure 15. The track class with the body and fields added is listed in Appendix B.

```
public class track implements java.io.Serializable
{
    private Integer trackNumber;

    public track(int inID){
        // insert the body here
    }
    public int getID(){
        // insert the body here
    }

    public void setPosition(position_type post){
        // insert the body here
    }

    public position_type getPosition(){
        // insert the body here
    }

    public String getCallsign(){
        // insert the body here
    }

    public void setCallsign(String sign){
        // insert the body here
    }

    // automatically generated do not delete!!
    public Integer autoGetID1(){
        return trackNumber ;
    }
}
```

Figure 15, An example of a distributed class generated by PSDL2Space

4. Object Wrapper

Wrapping is an approach to protecting legacy software systems and commercial off-the-shelf (COTS) software products that require no modification of those products [BER99]. It consists of two parts, an adapter that provides some additional functionality for an application program at key external interfaces, and an encapsulation mechanism that binds the adapter to the application and protects the combined components [BER99].

In this context, the software being protected contains the actual distributed objects, and the AICG model has no way of knowing the behaviors of the distributed object other than the type of operations of the methods. The adapter intercepts all invocations to provide additional functionalities such as synchronization between the local and distributed object, transaction control, events monitoring and exceptions handling. The encapsulation mechanism has been explained in the earlier section (AICG Architecture). Instead of instantiation of the actual object, the respective interface wrapper is instantiated. Instantiating the interface wrapper would indirectly instantiate the actual object as well as storing the object in the space.

Three classes are generated for every distributed object. These are named with the object name appended with the following Ext, ExtClient, and ExtServer. The class hierarchy of the example is shown in figure 16.

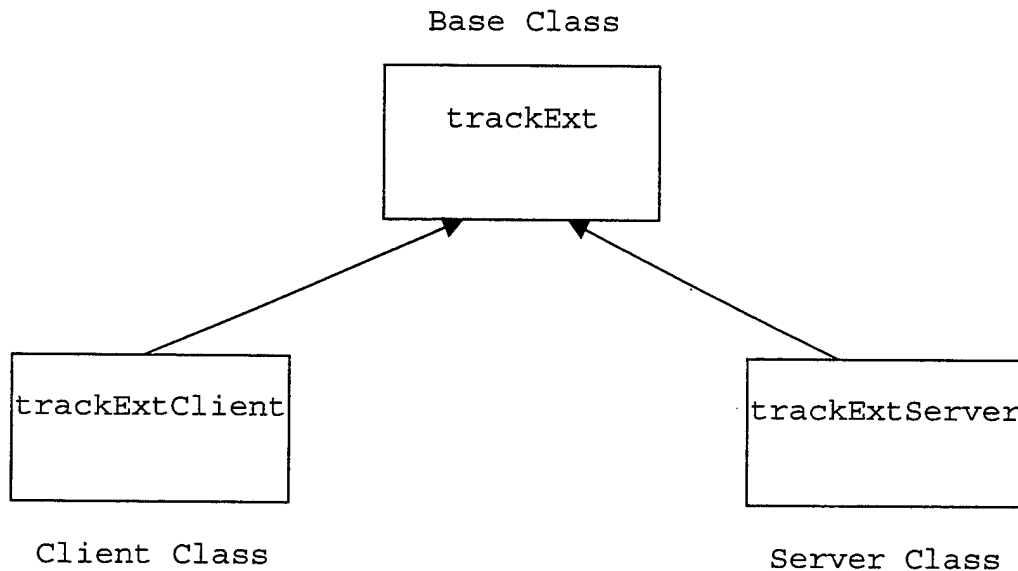


Figure 16, Wrapper Hierarchy

The trackExtClient class is initialized by applications that have only read access on the distributed object. Whereas, the trackExtServer class is initialized by applications that have both the read and write access on the distributed object. The listing of the three files are found in Appendix B.

C. EVENT MODULES

The event modules consist of the event callback template (notifyAICG), the event handler (evenAICGHandler) and the event identification object (eventAICGID).

1. Event Identification Class

The event identification class is used to distinguish one event from others. When an event of interest is registered, an event identification object is created by storing the identification and the event source. Together these two properties act to uniquely identify the event registration.

The class has only two methods, an 'equals' method that check if two event identification objects are the same and a 'to string' method which is used by the event handler for searching the right event objects from the hash table. The listing of the event identification class can be found in Appendix B.

2. Event Handler

Event Handler is the main body of the event operations in the AICG model. It handles registration of new events, deletion of old events, listening for event and invoking the right callback for that event. In fact, the event handler consists of three inner classes to perform the above functions. Events are stored in a hash table with the event

identification object as the key to the hash table. This allows fast retrieval of the event object and the respective callback methods.

The event handler listens for new events from the space or other sources. When an object is written to the space, an event is created by the space and captured by all the listeners. The event handler would immediately spawn a new thread and check whether the event is of interest to the application. Figure 17 shows the portion of the event handler that are executed when an event is received, the full listing of the module is found in Appendix B.

```
// call when an external event is "fired".
public void run() {
    // get the event id and source
    Object source = event.getSource();
    long id = event.getID();
    long seqN = event.getSequenceNumber();
    // create a new event identification object
    eventAICGID keyID= new eventAICGID(id,source);
    registerAICG tempReg;
    // create a key from the id and event source
    String key = new String(keyID.toString());
    // check if the key exist in the hash table (storage)
    if ((tempReg = (registerAICG) storage.get(key)) !=null)
    {
        // check if the event is an old or duplicate event
        if (seqN > tempReg.seqNum) {
            tempReg.seqNum = seqN;
            src.listenerAICGEvents(tempReg.anyObj);
        } else {
            // old events ignored
            return;
        }
    }
}
} // end of notifyHandler
} // end of class
```

Figure 17, Event Handler

3. The Callback Template

The callback template is a simple interface class with an abstract method `listenerAICGEvents`. Its main function is to allow the AICG model to invoke the application program when certain events of interest are "fired". As explained in section V.E, the template need to be implemented by the application that wishes to have notification. The interface abstract method need to be override by the application for the callback to work.

```
public interface notifyAICG
{
    // abstract method, to be override by the application
    // with the codes that are to execute when the event fired
    public abstract void listenerAICGEvents(Object obj);
}
```

Figure 18, notifyAICG interface

D. THE TRANSACTION MODULES

The transaction modules consist of transaction interface (`transactionAICG`) and the transaction factory (`transactionManagerAICG`).

The transaction interface class comprise of a group of static methods that are used for obtaining reference to the transaction manager server anywhere on the network. It uses

the Java RMI registry or the look-up server to locate the transaction server.

The transaction factory uses the transaction interface to obtain the reference to the server, which is then used to create the default transaction or user-defined transaction. In short, the transaction factory performs the following:

1. Invoke the transaction interface to obtain a transaction manager.
2. Create default transaction with lease time of 5 seconds.
3. Create transactions with user define lease time.

E. THE EXCEPTION MODULE

The exception module defines all the exception codes that are returned to the application when certain unexpected conditions occur in the AICG model. The exception include:

- "NotDefinedExceptionCode"; unknown error occur.
- "SystemExceptionCode"; system level exceptions, such disk failure, network failure.
- "ObjectNotFoundException"; the space does not contain the object.
- "TransactionException"; transaction server not found, transaction expire before commit.

- "LeaseExpireException"; object lease has expired.
- "CommunicationException"; space communication errors.
- "UnusableObjectException"; object corrupted.
- "ObjectExistException"; there another object with the same key in the space.
- "NotificationException"; events notification errors.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND RECOMMENDATION

A. CONCLUSIONS

This thesis demonstrates the ease of sharing distributed objects and automates the generation of generic interface wrappers directly from the Prototype System Description Languages. However, the design has a performance price penalty. Every read operation requires the interface to synchronize the local object with the distributed object before the value is returned. Every write operation requires two Space operations. Adding the overhead for transactions, event monitoring and control, reading operations are in the range of a hundred milliseconds and writing is in the range of a few hundred of milliseconds. The high overhead lies within the Java Virtual Machine (JVM), the JavaSpace Model and the network latency. Current versions of JVM and JavaSpace are in a premature state in terms of performance. Even so, the performances are still suitable for most applications that are not time critical. Similar implementations of distributed systems with the above features of AICG interface in CORBA and Java would not perform any better.

Performance is not the main concern regarding whether the AICG interface model is suitable for developing

interoperability between real-time applications, but predictability of the time and space of the operations are. Currently, the Java language lacks some important real-time programming features such as:

1. Language features for capturing common idioms for periodic activity and asynchronous event handling.
2. Predictability of time and space. The language must enable prediction of the worst-case execution time for language constructs, to avoid "priority inversion" and it should not run out of storage or fragment.

In the near future, if the JVM and the Java language embrace the Real-Time standards currently in final draft, it should be able to bridge the gap regarding requirements for real-time systems.

B. RECOMMENDATIONS

The AICG model is now still in the premature stages to be a useful developer tool. More features and developments are needed before the model becomes indispensable for developing distributed applications. The following sections describe some of the features that would enhance the usefulness of the AICG model.

1. Graphical User Interface (GUI)

A user-friendly Graphical Interface would allow users to develop applications using graphical objects to represent the interfaces and the distributed objects. The graphical editor would allow the developer to include operators, input and output similar to those of the CAPS system. An expert system would then provide the capability to generate the PSDL specifications for the graphical objects. This should speed up developments that use the AICG model.

2. Integration With The CAPS System

AICG model's primary objective is to allow developer to improve their productivity in developing reliable distributed systems by simplifying the interoperability between them.

The current version of CAPS, on the other hand, allows developers to design and build reliable end-systems. Integration of the AICG model with CAPS would complement each other, resulting in a total solution for developing truly distributed systems.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. PSDL GRAMMAR

1. psdl = {component}
2. component = data_type
 | operator
3. data_type = "type" id type_spec type_impl
4. type_spec = "specification" ["generic" type_decl]
 [type_decl]
 {"operator" id operator_spec}
 [functionality] "end"
5. operator = "operator" id operator_spec operator_impl
6. operator_spec = "specification" {interface}
 [functionality] "end"
7. interface = attribute [reqmts_trace]
8. attribute = "generic" type_decl
 | "input" type_decl
 | "output" type_decl
 | "states" type_decl "initially"
initial_expression_list
 | "exceptions" id_list
 | "maximum execution time" time
9. type_decl = id_list ":" type_name {"," id_list ":"
 type_name}
10. type_name = id | id "[" type_decl "]"
11. id_list = id {""," id}
12. reqmts_trace = "by requirements" id_list
13. functionality = [keywords] [informal_desc] [formal_desc]
14. keywords = "keywords" id_list
15. informal_desc = "description" "{" text "}"
16. formal_desc = "axioms" "{" text "}"

```

17. type_impl = "implementation ada" id "end"
    | "implementation" type_name {"operator" id
    operator_impl} "end"

18. operator_impl = "implementation ada" id "end"
    | "implementation" psdl_impl "end"

19. psdl_impl = data_flow_diagram [streams] [timers]
    [control_constraints]
    [informal_desc]

20. data_flow_diagram = "graph" {vertex} {edge}

21. vertex = "vertex" op_id [":" time]
    -- time is the maximum execution time

22. edge = "edge" id [":" time] op_id "->" op_id
    -- time is the latency

23. op_id = id ["(" [id_list] "|" [id_list] ")"]

24. streams = "data stream" type_decl

25. timers = "timer" id_list

26. control_constraints = "control constraints" constraint
    {constraint}

27. constraint = "operator" op_id
    ["triggered" [trigger] ["if" expression]
    [reqmts_trace]]
    ["period" time [reqmts_trace]]
    ["finish within" time [reqmts_trace]]
    ["minimum calling period" time [reqmts_trace]]
    ["maximum response time" time [reqmts_trace]]
    {constraint_options}

28. constraint_options = "output" id_list "if" expression
    [reqmts_trace]
    | "exception" id ["if" expression]
    [reqmts_trace]
    | timer_op id ["if" expression]
    [reqmts_trace]

29. trigger = "by all" id_list
    | "by some" id_list

```

```

30. timer_op = "reset timer"
           | "start timer"
           | "stop timer"

31. initial_expression_list = initial_expression {" ,"
initial_expression}

32. initial_expression = "true" | "false"
           | integer_literal | real_literal
           | string_literal | id
           | type_name "." id ["("
initial_expression_list ")"]
           | "(" initial_expression ")"
           | initial_expression binary_op
initial_expression
           | unary_op initial_expression

33. binary_op = "and" | "or" | "xor"
           | "<" | ">" | "=" | ">=" | "<=" | "/="
           | "+" | "-" | "&" | "*" | "/" | "mod" | "rem" |
***

34. unary_op = "not" | "abs" | "-" | "+"

35. time = integer_literal unit

36. unit = "microsec" | "ms"
           | "sec" | "min" | "hours"

37. expression_list = expression {" ," expression}

38. expression = "true" | "false"
           | integer_literal
           | time
           | real_literal
           | string_literal
           | id
           | type_name "." id ["(" initial_expression_list
")"]
           | "(" expression ")"
           | initial_expression binary_op initial_expression
           | unary_op initial_expression

39. id = letter {alpha_numeric}

40. real_literal = integer "." integer

```

```

41. integer_literal = digit {digit}
42. string_literal = "" {char} ""
43. char = any printable character except "}"
44. digit = "0 .. 9"
45. letter = "a .. z" | "A .. Z" | "_"
46. alpha_numeric = letter | digit
47. text = {char}

```

EXTENSION TO PSDL FOR AICG MODEL

Changes:

```

17. type_impl = "implementation ada" id "end"
      | "implementation" type_name {"operator" id
      operator_impl} "end"
      | "implemetation space" <space_impl>

18 <operator_impl> ::= implementation <id> <id> end
      | implementation <psdl_impl> end
      | implementation <class_impl> end

```

Additional:

```

51. <space_impl> ::= space {<property>} end

```

APPENDIX B. EXAMPLES OF GENERATED INTERFACE WRAPPER

ENTRYAICG.JAVA

```
//-----  
// Filename : entryAICG.java  
// Version : 0.1  
// Compiler : jdk 1.2 or J++  
// Generated: auto  
// Date : 01/07/00  
//-----  
  
/**  
 * Class autoCodeException  
 *  
 * @author Cheng Heng Ngom  
 * @version 0.1  
 *  
 * Description  
 * Basic components of the java space object. All distributed objects  
 * implements the entryAICG.  
 */  
package interfaceAICG;  
  
import net.jini.core.entry.Entry;  
  
public abstract class entryAICG implements Entry  
{  
    // main identification number  
    public Integer entryID;  
  
    // require by JavaSpace default constructor  
    public entryAICG( ) {  
    }  
  
    public entryAICG(int id){  
        entryID = new Integer(id);  
    }  
  
    // return the object stored in the entry  
    public abstract Object getObject( );  
  
}
```

TRACK.JAVA

```
//-----  
// Filename : track.java  
// Version  : 0.1  
// Compiler : jdk 1.2 or J++  
// Generated: auto  
// Date : 01/07/00  
//-----  
  
/**  
 *  
 * @author Cheng Heng Ngom  
 * @version 0.1  
 *  
 * Description  
 * Distributed Object, each object is distinguish by a unique  
 * tracknumber.  
 */  
package interfaceAICG;  
  
public class track implements java.io.Serializable  
{  
  
    private Integer tracknumber;  
    private String callsign;  
    private position_type position;  
  
    public track( int x) {  
        tracknumber = new Integer(x);  
        callsign =new String();  
        position = new position_type(0,0);  
    }  
  
    public int getID( ) {  
        return tracknumber.intValue();  
    }  
  
    public void setCallsign( String si) {  
        callsign = new String(si);  
    }  
  
    public String getCallsign( ) {  
        return callsign;  
    }  
  
    public void setPosition( position_type post) {  
        position.x = post.x;  
        position.y = post.y;  
    }  
  
    public position_type getPosition( ) {  
        return position;  
    }  
}
```

```

    }

    // automatically generated do not delete!!
    public synchronized Integer getaicgID1()
    {
        return tracknumber;
    }

}

```

TRACKENTRY.JAVA

```

//-----
// Filename : trackEntry.java
// Version   : 0.1
// Compiler  : jdk 1.2 or J++
// Generated: auto
// Date      : 01/07/00
//-----

/**
 *
 * @author Cheng Heng Ngom
 * @version 0.1
 *
 * Description
 *
 */
package interfaceAICG;

public class trackEntry extends entryAICG
{
    public Integer aicgID1;

    public track data;

    public trackEntry() {}

    public trackEntry(int aid, Integer aicgID1, track inData) {
        super(aid);
        data = inData;
        this.aicgID1 = aicgID1;
    }

    public Object getObject() {
        return data;
    }
}

```

TRACKEXT.JAVA

```
//-----  
// Filename : trackExt.java  
// Version  : 0.1  
// Compiler : jdk 1.2 or J++  
// Generated: auto  
// Date    : 01/07/00  
//-----  
  
/**  
 *  
 * @author Cheng Heng Ngom  
 * @version 0.1  
 *  
 * Description  
 *  
 */  
package interfaceAICG;  
  
import net.jini.core.lease.Lease;  
import net.jini.space.JavaSpace;  
import net.jini.core.event.*;  
  
public class trackExt implements java.io.Serializable  
{  
    protected JavaSpace space;  
    protected static transactionManagerAICG trnMgr;  
    protected eventAICGID notifyID;  
    protected eventAICGHandler evHandler;  
    protected boolean handlerEvent;  
    protected trackEntry distObj;  
    // user ids  
    protected Integer aicgID1;  
  
    /**  
     * Constructor, with notification disable  
     */  
    public trackExt()  
    {  
        throws exceptionAICG {  
            evHandler = null;  
            handlerEvent = false;  
            space = spaceAICG.getSpace(globalAICG.AICGSPACENAME);  
            // set the ids to null  
            aicgID1 = null;  
        }  
  
        // if there are n ids there will be n setID  
        public synchronized void setaicgID1( Integer inID)  
        {  
            aicgID1 = inID;  
        }  
    }  
}
```

```

    }

    // n set of get ids
    public synchronized Integer getaicgID1()
    {
        return aicgID1;
    }

    //always present xxxStratTxn
    public static void trackStartTxn()throws exceptionAICG{
        trnMgr = new transactionManagerAICG();
    }
}

```

TRACKEXTCLIENT.JAVA

```

//-----
// Filename : trackExtClient.java
// Version : 0.1
// Compiler : jdk 1.2 or J++
// Generated: auto
// Date : 01/07/00
//-----

/**
 *
 * @author Cheng Heng Ngom
 * @version 0.1
 *
 * Description
 *
 */
package interfaceAICG;

import net.jini.core.lease.Lease;
import net.jini.space.JavaSpace;

import net.jini.core.transaction.*;
import net.jini.core.transaction.server.*;

public class trackExtClient extends trackExt
{
    // client constructor
    public trackExtClient( Integer aicgID1) throws exceptionAICG{
        super();
        int retry = globalAICG.AICGDEFAULTRETRY;
        boolean trxSucc =false;
        Integer tempaicgID1 = aicgID1 ;
        while (retry !=0){
            try{
                // create atemplate to search for the object
                trackEntry template = new
trackEntry(globalAICG.AICGIDENTIFIER , tempaicgID1, null);

```

```

        distObj = (trackEntry) space.read(template,null,
            globalAICG.AICGDEFAULTWAITTIME);
        track ddata = (track) distObj.getObject();
        setaicgID1(ddata.getaicgID1());
        retry = 0;
        trxSucc = true;
    }catch (Exception e){
        System.err.println("retrying to find object");
        retry--;
    }
} // end of while
if (!trxSucc){
    throw new exceptionAICG(exceptionAICG.ObjectNotFoundException);
}
}

// event notification setting
public void setEvent(eventAICGHandler inEv) throws exceptionAICG
{
    evHandler = inEv;
    notifyID = evHandler.trackAddNotify(this);
}

// readonly methods
public int getID( ) throws exceptionAICG {
    trackEntry msgTemplate = new trackEntry(globalAICG.AICGIDENTIFIER,
    aicgID1, null);

    try{
        distObj = (trackEntry)
            space.read(msgTemplate, null,
            globalAICG.AICGDEFAULTWAITTIME);
        if (distObj==null){
            // object no longer exist
            throw new
            exceptionAICG(exceptionAICG.ObjectNotFoundException);
        }
        return distObj.data.getID();
    }catch (exceptionAICG ex){
        throw new exceptionAICG(exceptionAICG.ObjectNotFoundException);
    }catch (Exception e) {
        throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
    }
}

public String getCallsign( ) throws exceptionAICG {
    trackEntry msgTemplate = new trackEntry(globalAICG.AICGIDENTIFIER,
    aicgID1, null);

    try{
        distObj = (trackEntry)
            space.read(msgTemplate, null,
            globalAICG.AICGDEFAULTWAITTIME);
        if (distObj==null){
            // object no longer exist

```

```

        throw new
exceptionAICG(exceptionAICG.ObjectNotFoundException);
    }
    return distObj.data.getCallsign();
} catch (exceptionAICG ex) {
    throw new exceptionAICG(exceptionAICG.ObjectNotFoundException);
} catch (Exception e) {
    throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
}
}

public position_type getPosition() throws exceptionAICG {
    trackEntry msgTemplate = new trackEntry(globalAICG.AICGIDENTIFIER,
aicgID1, null);

    try{
        distObj = (trackEntry)
            space.read(msgTemplate, null,
globalAICG.AICGDEFAULTWAITTIME);
        if (distObj==null){
            // object no longer exist
            throw new
exceptionAICG(exceptionAICG.ObjectNotFoundException);
        }
        return distObj.data.getPosition();
    } catch (exceptionAICG ex) {
        throw new exceptionAICG(exceptionAICG.ObjectNotFoundException);
    } catch (Exception e) {
        throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
    }
}
}

```

TRACKEXTSERVER.JAVA

```

//-----
// Filename : trackExtServer.java
// Version : 0.1
// Compiler : jdk 1.2 or J++
// Generated: auto
// Date : 01/07/00
//-----

/**
 *
 * @author Cheng Heng Ngom
 * @version 0.1
 *
 * Description
 *
 */
package interfaceAICG;

import net.jini.core.lease.Lease;
import net.jini.core.space.JavaSpace;
import net.jini.core.transaction.*;
import net.jini.core.transaction.server.*;

```

```

public class trackExtServer extends trackExt
{
    // constructor
    public trackExtServer( int x) throws exceptionAICG{
        super();
        // create the template
        Transaction trn = null;
        try {
            trn = trnMgr.getDefaultTransaction();
            try{
                track ddata = new track(x);
                setaicgID1(ddata.getaicgID1());
                trackEntry template = new
trackEntry(globalAICG.AICGIDENTIFIER, aicgID1, null);
                //check if the distobj exit
                if (space.read(template, trn, JavaSpace.NO_WAIT) == null){
                    distObj = new trackEntry(globalAICG.AICGIDENTIFIER,
aicgID1, ddata);
                    space.write(distObj, trn, globalAICG.AICGLEASETIME);
                    trn.commit();
                }else{
                    // throws an exception
                    trn.abort();
                    throw new
exceptionAICG(exceptionAICG.ObjectExistException);
                }
            }catch (exceptionAICG e){
                trn.abort();
                throw new exceptionAICG(e.getErrorCode());
            }catch (Exception e) {
                trn.abort();
                e.printStackTrace();
                throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
            }
        }catch (exceptionAICG e) {
            throw new exceptionAICG(e.getErrorCode());
        }catch (Exception e) {
            throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
        }
    }

    // event notification setting
    public void setEvent(eventAICGHandler inEv)
    {
        evHandler = inEv;
    }

    // others methods
    public int getID( ) throws exceptionAICG {
        trackEntry msgTemplate = new trackEntry(globalAICG.AICGIDENTIFIER,
aicgID1, null);

        try{
            distObj = (trackEntry)

```

```

        space.read(msgTemplate, null,
globalAICG.AICGDEFAULTWAITTIME);
        if (distObj==null){
            // object nolonger exist
            throw new
exceptionAICG(exceptionAICG.ObjectNotFoundException);
        }
        return distObj.data.getID();
    }catch (exceptionAICG ex){
        throw new exceptionAICG(exceptionAICG.ObjectNotFoundException);
    }catch (Exception e) {
        throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
    }
}

public String getCallsign( ) throws exceptionAICG {
    trackEntry msgTemplate = new trackEntry(globalAICG.AICGIDENTIFIER,
aicgID1, null);

    try{
        distObj = (trackEntry)
            space.read(msgTemplate, null,
globalAICG.AICGDEFAULTWAITTIME);
        if (distObj==null){
            // object nolonger exist
            throw new
exceptionAICG(exceptionAICG.ObjectNotFoundException);
        }
        return distObj.data.getCallsign();
    }catch (exceptionAICG ex){
        throw new exceptionAICG(exceptionAICG.ObjectNotFoundException);
    }catch (Exception e) {
        throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
    }
}

public position_type getPosition( ) throws exceptionAICG {
    trackEntry msgTemplate = new trackEntry(globalAICG.AICGIDENTIFIER,
aicgID1, null);

    try{
        distObj = (trackEntry)
            space.read(msgTemplate, null,
globalAICG.AICGDEFAULTWAITTIME);
        if (distObj==null){
            // object nolonger exist
            throw new
exceptionAICG(exceptionAICG.ObjectNotFoundException);
        }
        return distObj.data.getPosition();
    }catch (exceptionAICG ex){
        throw new exceptionAICG(exceptionAICG.ObjectNotFoundException);
    }catch (Exception e) {
        throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
    }
}

```

```

public void setCallsign( String si) throws exceptionAICG {
int retry = globalAICG.AICGDEFAULTRETRY;
boolean trxSucc =false;
trackEntry msgTemplate =
    new trackEntry(globalAICG.AICGIDENTIFIER, aicgID1, null);
Transaction trn = null;

while(retry !=0){
    try {
        trn = trnMgr.createTransaction(globalAICG.USERTXNLEASE);
        try{
            distObj = (trackEntry)
                space.take(msgTemplate, trn, Long.MAX_VALUE);
            // called the actual method
            distObj.data.setCallsign(si);
            space.write(distObj,trn,globalAICG.AICGLEASETIME);
            trn.commit();
            retry = 0;
            trxSucc = true;
        }catch (Exception e){
            trn.abort();
            retry --;
            System.err.println("Exception level 1: retrying");
        }
    }catch (Exception e) {
        retry--;
        System.err.println("Exception level 2: retrying");
    }
} // end of while
if (!trxSucc){
    throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
}
}

public void setPosition( position_type post) throws exceptionAICG
{
    int retry = globalAICG.AICGDEFAULTRETRY;
    boolean trxSucc =false;
    trackEntry msgTemplate =
        new trackEntry(globalAICG.AICGIDENTIFIER, aicgID1, null);
    Transaction trn = null;

    while(retry !=0){
        try {
            trn = trnMgr.createTransaction(globalAICG.USERTXNLEASE);
            try{
                distObj = (trackEntry)
                    space.take(msgTemplate, trn, Long.MAX_VALUE);
                // called the actual method
                distObj.data.setPosition(post);
                space.write(distObj,trn,globalAICG.AICGLEASETIME);
                trn.commit();
                retry = 0;
                trxSucc = true;
            }catch (Exception e){
                trn.abort();
            }
        }
    }
}

```

```

        retry--;
        System.err.println("Exception level 1: retrying");
    }
    }catch (Exception e) {
        retry--;
        System.err.println("Exception level 2: retrying");
    }
    }// end of while
    if (!trxSucc){
        throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
    }
}
}

```

NOTIFYAICG.JAVA

```

//-----
// Filename : notifyAICG.java
// Version : 0.1
// Compiler : jdk 1.2 or J++
// Generated: auto
// Date : 01/07/00
//-----

/**
 * Class autoCodeException
 *
 * @author Cheng Heng Ngom
 * @version 0.1
 *
 * Description
 * interface class, user application must implements notifyAICG
 * for event callback to work.
 */
package interfaceAICG;

public interface notifyAICG
{
    public abstract void listenerAICGEvents(Object obj);
}

```

EVENTAICGID.JAVA

```

//-----
// Filename : eventAICGID.java
// Version : 0.1
// Compiler : jdk 1.2 or J++
// Generated: auto
// Date : 01/07/00
//-----

```

```

/**
 *
 * @author Cheng Heng Ngom
 * @version 0.1
 *
 * Description
 * event class, used to categorise the type of events
 */

package interfaceAICG;

import java.io.Serializable;

public class eventAICGID implements Serializable
{
    private long id;
    private Object src;

    public eventAICGID(long id, Object src){
        this.id=id;
        this.src=src;
    }

    public boolean equals(Object obj){
        if (obj == null){
            return false;
        }

        if (obj.getClass() != this.getClass()){
            return false;
        }

        eventAICGID tempID = (eventAICGID) obj;
        if ((id == tempID.id) && (src.equals(tempID.src))){
            return true;
        }else {
            return false;
        }
    }

    public String toString(){
        return (src.toString() + Long.toString(id));
    }
}

```

EVENTAICGHANDLER.JAVA

```

//-----
// Filename : eventAICGHandler.java
// Version : 0.1
// Compiler : jdk 1.2 or J++
// Generated: auto
// Date : 01/07/00

```

```

//-----
/**
 *
 * @author Cheng Heng Ngom
 * @version 0.1
 *
 * Description
 * Main event handler of the generated codes.
 */
package interfaceAICG;

import java.io.Serializable;
import java.util.Hashtable;

import java.rmi.RemoteException;
import java.rmi.server.*;
import net.jini.core.event.*;
import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import net.jini.core.transaction.server.*;
import net.jini.space.JavaSpace;

/**
 * eventAICGHandler, handles all the external notification from the
 * space
 */
public class eventAICGHandler implements RemoteEventListener,
Serializable
{
    private notifyAICG src;
    private Hashtable storage;
    private JavaSpace space;

    public class registerAICG implements Serializable{
        public EventRegistration reg;
        public long seqNum;
        public Object anyObj;

        public registerAICG(EventRegistration inReg, long inSeqNum,
Object obj){
            seqNum = inSeqNum;
            reg = inReg;
            anyObj= obj;
        }
    }

    public eventAICGHandler(notifyAICG inSrc) throws exceptionAICG{
        src = inSrc;
        space = spaceAICG.getSpace(globalAICG.AICGSPACENAME);
        storage = new Hashtable();
        // export the object
        try{
            UnicastRemoteObject.exportObject(this);
        }catch (RemoteException e){
            System.err.println("error in exporting object");
        }
    }
}

```

```

        e.printStackTrace();
        throw new exceptionAICG(exceptionAICG.NotificationException);
    }

}

public void notify(RemoteEvent event) {
    notifyHandler nh = new notifyHandler(event);
    new Thread(nh).start();
}

/**
 * trackAddNotify, add an object to the list of events to be
monitor
 */

    public eventAICGID trackAddNotify(trackExt indata) throws
exceptionAICG{
        EventRegistration reg;
        long seqNum;
        registerAICG tempReg;
        eventAICGID keyID;

        try{
            trackEntry template = new trackEntry(globalAICG.AICGIDENTIFIER
                                                ,indata.getaicgID1(),
null);
            reg = space.notify(template, null, this,
globalAICG.AICGLEASETIME,null);
            seqNum = reg.getSequenceNumber();
            tempReg = new registerAICG(reg,seqNum,indata);
            keyID = new eventAICGID(reg.getID(),reg.getSource());
            String key = new String(keyID.toString());
            storage.put(key,tempReg);

            return keyID;
        }catch (Exception e){
            System.err.println("Notification Exception");
            throw new exceptionAICG(exceptionAICG.NotificationException);
        }

    }

/**
 * trackAddNotify, add an object to the list of events to be
monitor with
 * transaction control
 */

    public eventAICGID trackAddNotify(trackExt indata, Transaction trn
)
        throws exceptionAICG{
        EventRegistration reg;
        long seqNum;
        registerAICG tempReg;
        eventAICGID keyID;

```

```

        try{
            trackEntry template = new trackEntry(globalAICG.AICGIDENTIFIER
                                                , indata.getaicgID1(),
null);
            reg = space.notify(template, trn, this, Lease.FOREVER,null);
            seqNum = reg.getSequenceNumber();
            tempReg = new registerAICG(reg,seqNum, indata);
            keyID = new eventAICGID( reg.getID(), reg.getSource());
            storage.put(keyID,tempReg);
            //
            System.out.println("Key: " + keyID);
            return keyID;
        }catch (Exception e){
            throw new exceptionAICG(exceptionAICG.NotificationException);
        }
    }
    /**
     * trackDeleteNotify, remove an object from the list of events to
be monitor
    */

    public void trackDeleteNotify(eventAICGID id) throws exceptionAICG{
        registerAICG tempReg;
        try {
            tempReg = (registerAICG) storage.remove(id);
            Lease lease = tempReg.reg.getLease();
            lease.cancel();

        }catch (Exception e){
            throw new exceptionAICG(exceptionAICG.NotificationException);
        }
    }

    public class notifyHandler implements Runnable{
        private RemoteEvent event;

        public notifyHandler(RemoteEvent event). {
            this.event = event;
        }

        public void run() {
            Object source = event.getSource();
            long id = event.getID();
            long seqN = event.getSequenceNumber();
            eventAICGID keyID= new eventAICGID(id,source);
            registerAICG tempReg;
            System.out.println("Key: " + keyID);
            String key = new String(keyID.toString());
            if ((tempReg = (registerAICG) storage.get(key)) !=null)
            {
                if (seqN > tempReg.seqNum) {
                    tempReg.seqNum = seqN;
                    src.listenerAICGEvents(tempReg.anyObj);
                } else {

                // old events ignored

```

```

        return;
    }
}
} // end of notifyHandler
} // end of class

```

TRANSACTIONAICG.JAVA

```

//-----
// Filename : transactionAICG.java
// Version : 0.1
// Compiler : jdk 1.2 or J++
// Generated: auto
// Date : 01/07/00
//-----

/**
 *
 * @author Cheng Heng Ngom
 * @version 0.1
 *
 * Description
 * static methods for supporting the transaction.
 */

package interfaceAICG;

import java.rmi.*;

import net.jini.core.transaction.server.TransactionManager;

import com.sun.jini.mahout.Locator;
import com.sun.jini.outrigger.Finder;

public class transactionAICG
{
    public static TransactionManager getManager(String name) {
        Locator locator = null;
        Finder finder = null;

        if (System.getSecurityManager() == null) {
            System.setSecurityManager(
                new RMISecurityManager());
        }

        if (System.getProperty("com.sun.jini.use.registry") !=
null) {
            locator = new com.sun.jini.mahout.RegistryLocator();
            finder = new com.sun.jini.outrigger.RegistryFinder();
        } else {
            System.out.println("lookup is used");
            locator = new
com.sun.jini.outrigger.DiscoveryLocator();
            finder = new com.sun.jini.outrigger.LookupFinder();
        }
    }
}

```

```

        return (TransactionManager)finder.find(locator, name);
    }

    public static TransactionManager getManager() {
        return
getManager(com.sun.jini.mahalo.TxnManagerImpl.DEFAULT_NAME);
    }
}

```

TRANSACTIONMANAGERAICG.JAVA

```

//-----
// Filename : transactionManagerAICG.java
// Version : 0.1
// Compiler : jdk 1.2 or J++
// Generated: auto
// Date : 01/07/00
//-----

/**
 * Class autoCodeException
 *
 * @author Cheng Heng Ngom
 * @version 0.1
 *
 * Description
 * Main transaction handler.
 */

package interfaceAICG;

import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import net.jini.core.transaction.server.*;

public class transactionManagerAICG
{
    private TransactionManager TxnMgr;

    public transactionManagerAICG() throws exceptionAICG {
        // get the reference to transaction Manager
        TxnMgr =
            transactionAICG.getManager();
    }

    public TransactionManager getTransactionManager() {
        return TxnMgr;
    }

    public Transaction getDefaultTransaction() throws exceptionAICG
    {
        Transaction defaultTxn;
        Transaction.Created trc = null;
        try {

```

```

        trc = TransactionFactory.create
            (TxnMgr,globalAICG.AICGDEFAULTTXNLEASE );
        defaultTxn = trc.transaction;
    } catch (Exception e) {
        System.err.println(
            "Could not create transaction ");
        throw new
exceptionAICG(exceptionAICG.TransactionException);
    }
    return defaultTxn;
}

    public Transaction createTransaction(long leaseTime)throws
exceptionAICG
    {
        Transaction Txn;
        Transaction.Created trc = null;
        try {
            trc = TransactionFactory.create(TxnMgr,leaseTime);
            Txn = trc.transaction;
            return Txn;
        } catch (Exception e) {
            System.err.println(
                "Could not create transaction ");
            throw new
exceptionAICG(exceptionAICG.TransactionException);
        }
    }
}

```

EXCEPTIONAICG.JAVA

```

//-----
// Filename : exceptionAICG.java
// Version : 0.1
// Compiler : jdk 1.2 or J++
// Generated : auto
// Date : 01/07/00
//-----

/**
 *
 * @author Cheng Heng Ngom
 * @version 0.1
 *
 * Description
 * Main Exception class of the automatic code generator. Each form of
 * excpetion is given an error code instead of having a new excetion
 * class define for it.
 */
package interfaceAICG;

public class exceptionAICG extends Exception

```

```

{
    /**
     * NotDefinedExceptionCode, given to any exception that
     * does not have an error code
     */
    public static final int NotDefinedExceptionCode = 0;

    /**
     * SystemExceptionCode
     */
    public static final int SystemExceptionCode = 1;

    /**
     * Object not found, time out exception
     */
    public static final int ObjectNotFoundException = 2;
    /**
     *
     */
    public static final int TransactionException = 3;
    /**
     *
     */
    public static final int LeaseExpireException = 4;
    /**
     *
     */
    public static final int CommunicationException = 5;

    /**
     *
     */
    public static final int UnusableObjectException = 6;

    /**
     *
     */
    public static final int ObjectExistException = 7;

    /**
     *
     */
    public static final int NotificationException = 8;

    public static final String[] ExceptionMessage=
    {"NotDefinedExceptionCode",
    "SystemExceptionCode",
    "ObjectNotFoundException",
    "TransactionException",
    "LeaseExpireException",

```

```

"CommunicationException",
"UnusableObjectException",
"ObjectExistException",
"NotificationException"};

/**
 * errorCode define the type of exception that occur
 */
public int errorCode;

/**
 * default constructor, no argument.
 */
public exceptionAICG(){
    super();
    errorCode = NotDefinedExceptionCode;
}

/**
 * Constructor.
 * @param code: int value of the error code
 */
public exceptionAICG(int code){
    super(ExceptionMessage[code]);
    errorCode = code;
}

/**
 * getErrorCode, return the erro code of the exception
 * @return int, error code
 */
public int getErrorCode(){
    return errorCode;
}
}

```

SPACEAICG.JAVA

```

//-----
// Filename : spaceAICG.java
// Version  : 0.1
// Compiler : jdk 1.2 or J++
// Generated: auto
// Date : 01/07/00
//-----

/**
 * Class autoCodeException
 */

```

```

* @author Cheng Heng Ngom
* @version 0.1
*
* Description
* static methods for supporting the java space.
*/

package interfaceAICG;

import java.rmi.*;

import net.jini.space.JavaSpace;

import com.sun.jini.mahout.binder.RefHolder;
import com.sun.jini.mahout.Locator;
import com.sun.jini.outrigger.Finder;

public class spaceAICG {

    public static JavaSpace getSpace(String name)throws exceptionAICG
    {
        try {
            if (System.getSecurityManager() == null) {
                System.setSecurityManager(
                    new RMISecurityManager());
            }

            if (System.getProperty("com.sun.jini.use.registry")
                == null)
            {
                Locator locator =
                    new com.sun.jini.outrigger.DiscoveryLocator();
                Finder finder =
                    new com.sun.jini.outrigger.LookupFinder();
                return (JavaSpace)finder.find(locator, name);
            } else {
                RefHolder rh = (RefHolder)Naming.lookup(name);
                return (JavaSpace)rh.proxy();
            }
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
        throw new exceptionAICG(exceptionAICG.SystemExceptionCode);
    }

    public static JavaSpace getSpace()throws exceptionAICG
    {
        return getSpace("DODSpaces");
    }
}

```

POSITION_TYPE.JAVA

```

package modelv2;
import java.io.Serializable;

```

```
public class position_type implements Serializable
{
    public int x;
    public int y;

    public position_type(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public String toString(){
        return (Integer.toString(x) + " " +Integer.toString(y));
    }
}
```

APPENDIX C. PSDL2SPACE CODE LISTINGS

AUTOMCODEEXCEPTION.JAVA

```
//-----  
// Filename : automCodeException.java  
// Version  : 1.0  
// Compiler : jdk 1.2 or J++  
//-----  
  
package psdl2java.global;  
/**  
 *  
 * @author Cheng Heng Ngom  
 * @version 1.0  
 *  
 * Description  
 * Main Exception class of the automatic code generator. Each form of  
 * excpetion is given an error code instead of having a new excetion  
 * class define for it.  
 */  
  
public class automCodeGenException extends Exception  
{  
    /**  
     * NotDefinedExceptionCode, given to any exception that  
     * does not have an error code  
     */  
    public static final int NotDefinedExceptionCode = -1;  
  
    /**  
     * SystemExceptionCode 0 to 9  
     */  
    public static final int SystemExceptionCode = 1;  
  
    /**  
     * Paser Exception is given the code 10 to 19  
     */  
    public static final int ParserExceptionCode = 10;  
  
    /**  
     * Wrong Type expected  
     */  
    public static final int ParserWrongTypeCode =11;  
  
    /**  
     * Syntax Error, eg, missing END  
     */  
    public static final int ParserSyntaxError = 12;  
  
    /**  
     * State or output parameter must not be of primitive type  
     */  
}
```

```

public static final int ParserStateError = 13;

/**
 * EOF detected, while in the middle of parser.
 */
public static final int ParserEOFExceptionCode = 14;

/**
 * ExceptionMessage, String of the exception in array for indexing.
 */
public static final String[]
    ExceptionMessage= {"NotDefinedExceptionCode",
                       null,null,null,null,null,
                       null,null,null,
                       "Parser: Exception Error",
                       "Parser: Wrong Type Code",
                       "Parser: Syntax Error",
                       "Parser: State Error",
                       "Parser: EOF detected"};

/**
 * errorCode defines the type of exception that occur
 */
public int errorCode;

/**
 * default constructor, no argument.
 */
public automCodeGenException() {
    super();
    errorCode = NotDefinedExceptionCode;
}

/**
 * Constructor.
 * @param code: int value of the error code
 */
public automCodeGenException(int code) {
    super();
    errorCode = code;
}

/**
 * getErrorCode, return the error code of the exception
 * @return int, error code
 */
public int getErrorCode() {
    return errorCode;
}

/**
 * printErrorCode, print the error code of the exception
 * @return void
 */
public void printErrorCode() {
    System.err.println("*** Exception Code: " + errorCode + " ")
}

```

```

        + ExceptionMessage[errorCode]);
    }
}

```

CLASSDEFINITION.JAVA

```

//-----
// Filename : classDefinition.java
// Version  : 1.0
// Compiler : jdk 1.2 or J++
//-----
/**
 * @author Cheng Heng Ngom
 * @version 1.0
 * classDefinition defines all the fields needed to
 * generate a class interface. It contains the name
 * of the server and the naming components.
 */
package psdl2java.global;
import java.util.Vector;
import java.io.Serializable;
import java.util.Enumuration;

public class classDefinition implements Serializable
{
    /**
     * distObjectName, name of the server
     */
    private String distObjectName;
    /**
     * method, stores all the name of the method
     */
    private Vector methods;
    /**
     * id, identification
     */
    private Vector id;
    /**
     * currMethod, current method of the class in use
     */
    private methodDefinition currMethod;
    /**
     * objectPro, object space properties
     */
    private spaceProperties objectPro;
    /**
     * distributed, is it an aicg object
     */
    private boolean distributed = true;
    /**
     * packageName, package name of the class
     */
}

```

```

private String packageName;
/**
 * count, number of identification id
 */
private int count = 0;

/**
 * Default Constructor, initialise all the values
 */
public classDefinition(){
    distObjectName = new String("");
    // create a new methods vector
    methods = new Vector();
    id = new Vector();
    objectPro = new spaceProperties();
    packageName = new String("");
}
/**
 * Constructor
 * @param sName, name of the class in Sting
 */
public classDefinition(String sName){
    distObjectName = new String(sName);
    // create a new methods vector
    methods = new Vector();
    id = new Vector();
    objectPro = new spaceProperties();
    packageName = new String("");
}

/**
 * setName, set the class name
 * @param name, new name in String
 * @return void
 */
public void setName(String name) {
    distObjectName = new String (name);
}

/**
 * getName, return the name in String
 * @return String
 */
public String getName (){
    return distObjectName;
}

/**
 * addMethod, add a method to the class
 * @param sName, name of the method in String
 * @param sretType, return type in String
 * @return methodDefinition, method class
 */
public methodDefinition addMethod(String sName, String sretType){
    currMethod = new methodDefinition(sName, sretType);
    // add a new method into the vector

```

```

        methods.addElement(currMethod);
        return currMethod;
    }
    /**
     * addMethod, add a method to the class with return type as void
     * @param sName, name of the class in String
     * @return methodDefinition, method class
     */
    public methodDefinition addMethod(String sName){
        currMethod = new methodDefinition(sName);
        // add a new method into the vector
        methods.addElement(currMethod);
        return currMethod;
    }

    /**
     * addID, add a new identification variable
     * @param sName, name of the variable
     * @param ty, type of the variable in String
     * @return void
     */
    public void addID(String sName, String ty){
        variableDefinition vd = new
            variableDefinition(definition.IN,sName,ty);
        id.addElement(vd);
        // refresh the ID string
        vd.mapID();
        getIDTypes();
        count ++;
    }

    /**
     * getIDs, return a list of identification IDs
     * @return Vector, list of Ids
     */
    public Vector getIDs(){
        return id;
    }

    /**
     * getMethod, return a list of methods
     * @return Vector, list of Method Definition
     */
    public Vector getMethods(){
        return methods;
    }
    /**
     * getCurrentMethod, return the current method
     * @return methodDefinition
     */
    public methodDefinition getCurrentMethod(){
        return currMethod;
    }
    /**
     * getProperties, return the space properties of the class
     * @return spaceProperties

```

```

    */
    public spaceProperties getProperties() {
        return objectPro;
    }
    /*
    * setObjectState, set the space property distributed
    * @param state, new distribute state in boolean
    * @return void
    */
    public void setObjectState(boolean state) {
        distributed = state;
    }
    /*
    * getObjectState, return the space distributed state
    * @return boolean, the state of the object (true -use space, false
    * normal object)
    */
    public boolean getObjectState() {
        return distributed;
    }
    /*
    * setPackageName, set the name of the package the class belong to.
    * @param name, name of the package
    * @return void
    */
    public void setPackageName(String name) {
        packageName = name;
    }
    /*
    * getPackageName, return the package name the class belong to.
    * @return String, name of the package
    */
    public String getPackageName() {
        return packageName;
    }
    /**
    * getIDTypes, return the type follow by name of the ID in String
    * @return String, eg. int aicgID1, ...
    */
    public String getIDTypes() {
        //
        Enumeration enumIDs = id.elements();
        variableDefinition vDef;
        String sTemp = null;
        if (enumIDs.hasMoreElements()) {
            vDef = (variableDefinition) enumIDs.nextElement();
            sTemp = new String(vDef.getType() + ' ' +
                               vDef.getClassID().getAICGName());
            while (enumIDs.hasMoreElements()) {
                vDef = (variableDefinition) enumIDs.nextElement();
                sTemp = new String(sTemp + ", " + vDef.getType() + ' ' +
                                   vDef.getClassID().getAICGName());
            }
        }
    }

```

```

    }
    }
    return sTemp;

}
/**
 * getIDNames, return only the ID names separated by comma
 * @return String, eg. aicgID1, aicgID2, etc
 */
public String getIDNames(){
    Enumeration enumIDs = id.elements();
    variableDefinition vDef;
    String sTemp = null;
    if (enumIDs.hasMoreElements()){
        vDef = (variableDefinition) enumIDs.nextElement();
        sTemp = new String(vDef.getClassID().getAICGName());
        while (enumIDs.hasMoreElements()){
            vDef = (variableDefinition) enumIDs.nextElement();
            sTemp = new String(sTemp + ", " +
                               vDef.getClassID().getAICGName());
        }
    }
    return sTemp;
}

/**
 * getNoIDs, return the number of ID in this class
 * @return int, number of ID
 */
public int getNoIDs(){
    return count;
}

}

```

CLASSIDDEFINITION.JAVA

```

//-----
// Filename : classIDDefinition.java
// Version  : 1.0
// Compiler : jdk 1.2 or J++
//-----

package psdl2java.global;

/**
 * @author Cheng Heng Ngom
 * @version 1.0
 * Description
 * Defines the identification structure.
 */
public class classIDDefinition extends Object {
    /**
     * vD, variable class of the Id

```

```

    */
private variableDefinition vD;
/**
 * aicgIDName, local name of the ID, used by the AICG only
 */
private String aicgIDName;
/**
 * count, local counter.
 */
static private int count = 1;

/**
 * default constructor.
 * @param inVD, input variable Definition.
 */
public class IDDefinition(variableDefinition inVD) {
    aicgIDName = new String("aicgID" + count);
    vD = inVD;
    count ++;
}
/**
 * getAICGName, return the local ID name
 * @return String, name of the local ID
 */
public String getAICGName() {
    return aicgIDName;
}

/**
 * getVD, return the variable Definition of the class
 * @return variableDefinition.
 */
public variableDefinition getVD() {
    return vD;
}

```

DEFINITION.JAVA

```

/-----
// Filename : definition.java
// Version  : 1.0
// Compiler : jdk 1.2 or J++
/-----
/**
 * @author Cheng Heng Ngom
 * @version 1.0
 * definition defines all the global static fields
 * of the AICG parser and generator
 */
package psdl2java.global;

public class definition
{
    public final static int MAXNUMBERMETHODS = 100;
    public final static int MAXRESERVEWORDS = 100;
}

```

```

public final static int MAXTOKEN = 31;
public final static int MAXFIX = 31;

// for future ada implemetation-- not used for java
public final static int IN = 0;
public final static int OUT = 1;
public final static int INOUT = 2;

// java primitive type
public final static int SBOOLEAN = 1;
public final static int SCHAR = 2;
public final static int SBYTE = 3;
public final static int SSHORT = 4;
public final static int SINT = 5;
public final static int SLONG = 6;
public final static int SFLOAT = 7;
public final static int SDOUBLE = 8;

//java non-primitive type
public final static int SSTRING = 9;
public final static int VECTOR = 11;
// java undeclare type
public final static int SVOID = 10;

// user define type
public final static int SUSERTYPE = 0;

// Java object for primitive type
public final static String[] SUBJECT= { null, "Boolean",
"Character",
"Byte", "Short",
"Integer", "Long", "Float",
"Double", "String",
"void"};
// java primitive type
public final static String[] SPRIMITIVE = { null, "boolean",
"char",
"byte", "short",
"int", "long", "float",
"double", "String",
"void"};

// SPACE method properties
public final static int SPCONSTRUCTOR = 0;
public final static int SPREAD = 1;
public final static int SPWRITE = 2;
public final static long SPTTIME = 2000;

public final static int SPYES = 1;
public final static int SPNO = 0;
public final static int SPSERVER = 0;
public final static int SPSERVERS = 1;
public final static int SPALL = 2;

```

```

public final static int SPMANY = 11;
public final static int SPONE = 10;

// default setting
public static String DEFAULTPACKAGENAME = "interfaceAICG";
public static final String DEFAULTSPACENAME = "DODSpaces";

/**
 * spaceTranslate, translate a String name into the
 * AICG enumerate type
 * @param name, String of the type
 * @return int, the ennumerate type
 */
public static int spaceTranslate(String name)
throws automCodeGenException{
    if (name.compareTo("CONSTRUCTOR")== 0){
        return SPCONSTRUCTOR;
    }else if (name.compareTo("READ")== 0){
        return SPREAD;
    }else if (name.compareTo("WRITE")== 0){
        return SPWRITE;
    }else if (name.compareTo("YES")== 0){
        return SPYES;
    }else if (name.compareTo("NO")== 0){
        return SPNO;
    }else if (name.compareTo("SERVER")== 0){
        return SPSERVER;
    }else if (name.compareTo("SERVERS")== 0){
        return SPSERVERS;
    }else if (name.compareTo("ALL")== 0){
        return SPALL;
    }else if (name.compareTo("MANY")== 0){
        return SPMANY;
    }else if (name.compareTo("ONE")== 0){
        return SPONE;
    }else{
        throw new automCodeGenException(
            automCodeGenException.ParserWrongTypeCode);
    }
}

}

```

METHODDEFINITION.JAVA

```

//-----
// Filename : methodDefinition.java

```

```

// Version : 0.2
// Compiler : jdk 1.2 or J++
//-----
/**
 * @author Cheng Heng Ngom
 * @version 0.1
 * methodDefinition defines the method and it parameters
 *
 */
package psdl2java.global;
import java.io.Serializable;
import java.util.Vector;
import java.util.Enumuration;

public class methodDefinition implements Serializable
{
    /**
     * paramters, input and output fields of the method
     */
    private Vector parameters;
    /**
     * name, name of the method
     */
    private String name;
    /**
     * returnType, return type in String
     */
    private String returnType;
    /**
     * enumeration type of the return type
     */
    private int rtype;
    /**
     * spacemode, mode :SPREAD, SPWRITE, SPCONSTRUCTOR
     */
    private int spaceMode = definition.SPREAD;
    /**
     * spaceTransaction, define the transaction control on
     * the method.(SPYES, SPNO)
     */
    private int spaceTransaction = definition.SPNO;
    /**
     * spaceTransactionTime, define the time the transaction must
     * commit before it expire.( nnnn ms, deaault 2000ms)
     */
    private long spaceTransactionTime = definition.SPTTIME;

    /**
     * Constructor.
     * @param mname, name of the method
     */
    public methodDefinition(String mname){
        parameters = new Vector();
        name = new String(mname);
        returnType = new String("void");
        rtype = variableDefinition.javaType("void");
    }
}

```

```

/**
 * Constructor.
 * @param mname, name of the method
 * @param returnValue, return type
 */
public methodDefinition(String mname, String returnValue){
    parameters = new Vector();
    name = new String(mname);
    returnType = new String (returnValue);
    rtype = variableDefinition.javaType(returnValue);
}

/**
 * setRetType, modifies the return type of the method
 * @param returnValue, return type in String
 * @return void
 */
public void setRetType(String returnValue){
    rtype = variableDefinition.javaType(returnValue);
    if (rtype == definition.SUSERTYPE){
        returnType = new String(returnValue);
    }else {
        returnType = new String(definition.SPRIMITIVE[rtype]);
    }
}

/**
 * getRetType, return the return type of the method
 * @return String, return type in String
 */
public String getRetType(){
    return returnType;
}

/**
 * getJavaRetType, return the return type in Java object enumerate
 * type
 * @return int, enumerate type in int
 */
public int getJavaRetType(){
    return rtype;
}

/**
 * addParameter, add new input.output parameter into the method
 * @param dir, input, output or both
 * @param n, name of the variable
 * @param ty, type of the variable
 * @return void
 */
public void addParameter(int dir, String n, String ty){
    variableDefinition vd = new variableDefinition(dir,n,ty);
    parameters.addElement(vd);
}

/**

```

```

    * getName, return the name of the methods
    * @return String, name of the method
    */

    public String getName() {
        return name;
    }

    /**
     * getParameters, return the list of parameter
     * @return Vector, list of variableDefinition
     */
    public Vector getParameters() {
        return parameters;
    }

    /**
     * getParametersString, return the list of parameter
     * @return String, list of variableDefinition in the form
     * Integer name, Float etc
     */
    public String getParameterString() {
        Enumeration enumIDs = parameters.elements();
        variableDefinition vDef;
        String sTemp = new String();
        int counter = 0;
        if (enumIDs.hasMoreElements()) {
            vDef = (variableDefinition) enumIDs.nextElement();
            sTemp = new String(vDef.toString());
            while (enumIDs.hasMoreElements()) {
                vDef = (variableDefinition) enumIDs.nextElement();
                if (counter == 0) {
                    sTemp = new String(vDef.toString());
                } else {
                    sTemp = new String(sTemp + ", " + vDef.toString());
                }
                counter++;
            }
        }
        return sTemp;
    }

    /**
     * getParameterName, return the list of parameter names of the
    method
     * @return String, the name of the parameters in aicgID1, aicgID2
    ...
    */
    public String getParameterName() {
        Enumeration enumIDs = parameters.elements();
        variableDefinition vDef;
        String sTemp = new String();
        if (enumIDs.hasMoreElements()) {
            vDef = (variableDefinition) enumIDs.nextElement();
            sTemp = new String(vDef.getID());
            while (enumIDs.hasMoreElements()) {
                vDef = (variableDefinition) enumIDs.nextElement();

```

```

        sTemp = new String(sTemp + ", " + vDef.getID());
    }
    }
    return sTemp;
}

/**
 * setReadProperty, set the method whether it is
 * a read, or write.
 * @param int, 0 : read, 1: write
 * @return void
 */
public void setReadProperty(int mode){
    spaceMode = mode;
    if ( spaceMode == definition.SPConstructor){
        returnType = new String();
    }
}

/**
 * getReadProperty, return the read property
 * @return int, 0: constructor, 1:read, 2: write
 */
public int getReadProperty(){
    return spaceMode;
}

/**
 * setTransaction, set whether transaction is enable for this
method
 * @return void
 */
public void setTransaction(int ty){
    spaceTransaction = ty;
}

/**
 * getTransaction, return the value of transaction
 * @ return int, SPYES : transaction enable, SPNO: transaction
disable
 */
public int getTransaction(){
    return spaceTransaction;
}

/**
 * setTransactionTime, set the transaction time before it expire
 * @param ty, time in minisecond
 * @return void
 */
public void setTransactionTime(long ty){
    spaceTransactionTime = ty;
}

/**
 * getTransactionTime, return the transaction time of the method
 * @return long, the transaction time in miniseconds
 */
public long getTransactionTime(){
    return spaceTransactionTime;
}

```

}

}

READER.JAVA

```
//-----  
// Filename : reader.java  
// Version : 1.0  
// Compiler : jdk 1.2 or J++  
//-----  
  
package psdl2java.global;  
  
import java.io.*;  
  
/**  
 * Class reader  
 *  
 * @author Cheng Heng Ngom  
 * @version 0.1  
 *  
 * Description  
 * An abstract class that get the reserve words from a file and  
 * set up the tokenizer for the input file  
 */  
  
public abstract class reader  
{  
    /**  
     * arrays of reserve words  
     */  
    protected String reserveWords[];  
  
    /**  
     * Reader file to obtain the input to be parsed  
     */  
    private BufferedReader inReader;  
  
    /**  
     * Reader file to obtain the reserve words of the languages  
     */  
    private BufferedReader reserveFileReader;  
  
    /**  
     * Input Stream Token  
     */  
    protected StreamTokenizer inStreamTokenizer;  
  
    /**  
     * Arguments Constructor,  

```

```

* @param inFileName: input PSDL file
* @param reserveFileName: reserve words text file
*/
public reader(String inFileName, String reserveFileName)
throws automCodeGenException{
    // check if the file exist
    try{
        inReader = new BufferedReader( new FileReader(inFileName));
        reserveFileReader = new BufferedReader( new
FileReader(reserveFileName));

        }catch (IOException e){
            System.err.println("File open error");
            throw new
automCodeGenException(automCodeGenException.ParserExceptionCode);
        }
        reserveWords = new String[definition.MAXRESERVEWORDS];

        try{
            // get the reserve words
            getReserve();

            // initialise the stream tokenizer
            initStreamTokenizer();

        }catch (automCodeGenException e){
            throw new automCodeGenException(e.errorCode);
        }
    }
}
/**
* getReserve, gets the reserve words from a file and save it on
* an array.
* @return void
* @exception automCodeGenException
*/
private void getReserve()throws automCodeGenException {
    int srType;
    int i =0;
    try {
        StreamTokenizer sReserve = new
StreamTokenizer(reserveFileReader);
        // ignore the EOL
        sReserve.eolIsSignificant(false);
        sReserve.ordinaryChar(95);
        srType = sReserve.nextToken();

        while (srType != StreamTokenizer.TT_EOF){
            if (srType == StreamTokenizer.TT_WORD){
                reserveWords[i] = new String (sReserve.sval);
                i++;
                srType = sReserve.nextToken();
            }
        }
    } // end of while
} catch (Exception e){
    throw new automCodeGenException(
        automCodeGenException.ParserExceptionCode);
}

```

```

    }
}

/**
 * initStreamTokenizer, provide basic initialization of the input
 * tokens. should be overridden for customization.
 * @return void;
 * @exception automCodeException
 */

public void initStreamTokenizer() throws automCodeGenException
{
    inStreamTokenizer = new StreamTokenizer(inReader);
    inStreamTokenizer.eolIsSignificant(true);

    // any other initialise needed?
}

/**
 * Abstract method to be implemented by the sub class on how the
 * tokens is used
 */
public abstract String getNextToken(boolean checkEOL) throws
automCodeGenException;

/**
 * Abstract method to define the main production of the grammars.
 */
public abstract boolean mainProduction() throws
automCodeGenException;

} // end of class

```

SPACEPROPERTIES.JAVA

```

//-----
// Filename : spaceProperties.java
// Version  : 1.0
// Compiler : jdk 1.2 or J++
//-----
/**
 * @author Cheng Heng Ngom
 * @version 1.0
 */

package psdl2java.global;

import java.io.Serializable;

/**
 * spaceProperties defines all the fields for the AICG
 * space generated interface.

```

```

*
*/
public class spaceProperties implements Serializable
{
    /**
     * spaceName, name of the space to use
     */
    private String spaceName;
    /**
     * ownership, allows server to use the object
     */
    private int ownership = definition.SPYES;
    /**
     * security, type of security; server/all
     */
    private int security = definition.SPSERVER;
    /**
     * lease, time in ms before the object expired
     */
    private long lease = Long.MAX_VALUE ;
    /**
     * clone, duplication allow
     */
    private int clone = definition.SPONE;
    /**
     * notify, notication of modification enable/disable
     */
    private int notify = definition.SPNO;
    /**
     * constructor, create a new space properties with
     * default setting.
     */
    public spaceProperties(){
        spaceName= new String("DODSpaces");

        ownership = definition.SPYES;

        security = definition.SPSERVER;

        lease = Long.MAX_VALUE;

        clone = definition.SPONE;

        notify = definition.SPNO;
    }
    /**
     * Argument constructor
     * @param name, Name of the Space
     * @param own, ownership
     * @param sec, security
     * @param duration, lease time in ms
     * @param num, clone number
     * @param notifyIn, notification enable or disable
     */
    public spaceProperties(String name, int own, int sec, int duration,
        int num, int notifyIn){

```

```

        spaceName= new String(name);

        ownership = own;

        security = sec;

        lease = duration;

        clone = num;

        notify = notifyIn;
    }
    /**
     * setName, change the name of the space
     * @param name, name of the new space
     * @return void
     */
    public void setName(String name){
        spaceName = name;
    }
    /**
     * getName return the name of the space
     * @return String, name of the space
     */
    public String getName(){
        return spaceName;
    }
    /**
     * setOwnership, set the ownership properties
     * @param own, new ownership
     * @return void
     */
    public void setOwnership(int own){
        ownership = own;
    }
    /**
     * getOwnership, return the ownership value
     * @return int, ownership value
     */
    public int getOwnership(){
        return ownership;
    }
    /**
     * setSecurity, set the security property
     * @param indata, security value
     * @return void
     */
    public void setSecurity(int indata){
        security = indata;
    }
    /**
     * getSecurity, return teh security value
     * @return int, security value
     */
    public int getSecurity(){
        return security;
    }

```

```

/**
 * setLease, set the lease in ms
 * @param indata, new lease in ms
 * @return void
 */
public void setLease(long indata){
    lease = indata;
}
/**
 * getLease, return the lease in ms
 * @return long, lease in ms
 */
public long getLease(){
    return lease;
}
/**
 * setClone, set the number of clone allow
 * @param indata, number of clone allow
 * @return void
 */
public void setClone(int indata){
    clone = indata;
}
/**
 * getClone, return the number of clone
 * @return int, number of clone allow
 */
public int getClone(){
    return clone;
}
/**
 * setNotify, set the notify for modification
 * @param indata, value of the notify
 * @return void
 */
public void setNotify(int indata){
    notify = indata;
}
/**
 * getNotify, return the value of notify
 * @return int, value of notify
 */
public int getNotify(){
    return notify;
}
}

```

VARAIBLEDEFINITION.JAVA

```

//-----
// Filename : variableDefinition.java
// Version  : 1.0
// Compiler : jdk 1.2 or J++
//-----

```

```

package psdl2java.global;

import java.io.Serializable;

/**
 * @author Cheng Heng Ngom
 * @version 0.1
 * variableDefinition defines a type definition.
 * input/ouput type_id : type
 */

public class variableDefinition implements Serializable
{
    /**
     * io, state whether the variable is an input or output
     */
    private int io = definition.IN;
    /**
     * varID, name of the variable
     */
    private String varID;
    /**
     * stype, type of the variable in String
     */
    private String stype;
    /**
     * itypem type of variable in integer enumerated
     */
    private int itype;
    /**
     * cID, keep track of the aicg names used in conjunction with this
     * variable
     */
    private classIDDefinition cID;

    /**
     * Default Constructor.
     * @param sio, new input or output state
     * @param id, name of the variable
     * @param ty, type of the variable
     */
    public variableDefinition(int sio, String id, String ty) {
        io = sio;
        varID = new String(id);
        itype = javaType(ty);
        if (itype == definition.SUSERTYPE) {
            stype = new String(ty);
        } else {
            stype = new String(definition.SPRIMITIVE[itype]);
        }
    }

    /**
     * getID, return the name of the variable

```

```

    * @return String, name of the variable
    */
    public String getID(){
        return varID;
    }

    /**
     * getType, return the type of the variable
     * @return String, type of the variable
     */
    public String getType(){
        return stype;
    }

    /**
     * getIType, return the type of the variable in integer
     * enumeration.
     * @return int, type of the variable in int enumeration
     */
    public int getIType(){
        return itype;
    }

    /**
     * javaType, convert the String type into int enumeration
     * type.
     * @param stype, type in String.
     * @return int, enumeration type of the input String
     */
    public static int javaType(String stype){
        // java primitive type
        if (stype.compareTo("boolean")==0){
            return definition.SBOOLEAN;
        }else if (stype.compareTo("char")==0){
            return definition.SCHAR;
        }else if (stype.compareTo("byte")==0){
            return definition.SBYTE;
        }else if (stype.compareTo("short")==0){
            return definition.SSHORT;
        }else if (stype.compareTo("integer")==0){
            return definition.SINT;
        }else if (stype.compareTo("long")==0){
            return definition.SLONG;
        }else if (stype.compareTo("float")==0){
            return definition.SFLOAT;
        }else if (stype.compareTo("double")==0){
            return definition.SDOUBLE;
        }else if (stype.compareTo("string")==0){
            return definition.SSTRING;
        }else if (stype.compareTo("void")==0){
            return definition.SVOID;
        }else {
            return definition.SUSERTYPE;
        }
    }

    /**

```

```

    * getJavaObject, return the type convert to Java Object in String
    * @return String, type of the variable in Java Object
    */
    public String getJavaObject(){
        return definition.SUBJECT[itype];
    }

    /**
     * toString, override the base method.
     * @return String, variable declaration format
     * eg. Integer id1
     */
    public String toString(){
        return stype + ' ' + varID;
    }

    /**
     * mapID, create a new classIDDefinition for this object
     * @return void
     */
    public void mapID(){
        CID = new classIDDefinition(this);
    }

    /**
     * getClassID, return the classIDDefinition create for this object
     * @return classIDDefinition, ClassIDObject for this object
     */
    public classIDDefinition getClassID(){
        return CID;
    }
}

```

PSDLDEFINITION.JAVA

```

/-----
// Filename : psdlDefinition.java
// Version : 1.0
// Compiler : jdk 1.2 or J++
//-----

package psdl2java.parser;
/**
 * @author Cheng Heng Ngom
 * @version 1.0
 * psdlDefinition interface defines al the constants used
 * by the AICG.
 */
public interface psdlDefinition
{
    final int TYPE = 0;
    final int OPERATOR = 1;
    final int SPECIFICATION = 2;
    final int IMPLEMENTATION = 3;
    final int GRAPH = 4;
    final int EDGE = 5;
    final int STATES = 6;
}

```

```

final int INPUT = 7;
final int OUTPUT = 8;
final int PROPERTY = 9;
final int AXIOMS = 10;
final int VERTEX = 11;
final int GENERIC = 12;

final int IF = 13;
final int MS = 14;
final int SEC = 15;
final int END = 16;
final int MIN = 17;
final int MICROSEC = 18;
final int KEYWORDS = 19;

final int TRIGGERED = 20;
final int EXCEPTION = 21;
final int INITIALLY = 22;
final int EXCEPTIONS = 23;
final int DESCRIPTION = 24;
final int OR = 25;
final int AND = 26;
final int MOD = 27;
final int REM = 28;
final int XOR = 29;
final int DIGIT = 30;
final int LETTER = 31;
final int TRUE = 32;
final int FALSE = 33;
final int TIMER = 34;
final int HOURS = 35;
final int PERIOD = 36;
final int EOL = 37;
// added new Definition

final int CORBA = 38;
final int JAVA = 39;
final int SPACE = 40;
final int SPACEMODE = 41;
final int READ = 42;
final int WRITE = 43;
final int CONSTRUCTOR = 44;
final int SPACENAME = 45;
final int SPACELEASE = 46;
final int SERVER = 47;
final int EXTENDED = 48;
final int SPACESECURITY = 49;
final int SERVERS = 50;
final int ALL = 51;
final int OWNERSHIP = 52;
final int SECURITY = 53;
final int LEASE = 54;
final int CLONE = 55;
final int NOTIFY = 56;
final int TRANSACTION = 57;
final int TRANSACTIONTIME = 58;

```

```
}
```

PSDLREADER.JAVA

```
//-----  
// Filename : psdlReader.java  
// Version  : 1.0  
// Compiler : jdk 1.2 or J++  
//-----  
  
package psdl2java.parser;  
  
import java.io.*;  
import java.util.Vector;  
  
import psdl2java.global.*;  
/**  
 * @author Cheng Heng Ngom  
 * @version 1.0  
 * psdlReader implement the reader which check the  
 * syntax of the PSDL input file and extract the necessary  
 * information  
 */  
public class psdlReader extends reader implements psdlDefinition  
{  
    /**  
     * Name of the server  
     */  
    private Vector distObject;  
  
    /**  
     * class properties of the interface  
     */  
    private classDefinition mainServerClass;  
  
    /**  
     * Argumentals constructor.  
     * @param inFileName, name of the PSDL input file  
     * @param reserveFileName, name of the PSDL reserve file name  
     */  
    public psdlReader(String inFileName, String reserveFileName)  
        throws automCodeGenException{  
  
        super(inFileName, reserveFileName);  
        // initialise the distObject list  
        distObject = new Vector();  
        //include ':' as a white space  
        inStreamTokenizer.whitespaceChars(58, 58);  
        //include '=' as a white space  
        inStreamTokenizer.whitespaceChars('=', '=');  
        // include '_' as a character  
        inStreamTokenizer.wordChars(95, 95);  
    }  
}
```

```

    }

    /**
     * getNextToken, get the next token from the input stream
     * @param checkEOL, if true, EOL is treaded as a token else ignored
     * @return String, the token
     */
    public String getNextToken(boolean checkEOL) throws
    automCodeGenException{

        try{
            while (true){
                inStreamTokenizer.nextToken();
                if (inStreamTokenizer.ttype != StreamTokenizer.TT_EOF){
                    if (inStreamTokenizer.ttype == StreamTokenizer.TT_WORD)
                    {
                        return inStreamTokenizer.sval;
                    }else if (inStreamTokenizer.ttype ==
StreamTokenizer.TT_EOL)
                    {
                        if (checkEOL){
                            return null;
                        }
                    }else throw new automCodeGenException(
                        automCodeGenException.ParserWrongTypeCode);
                    }else {
                        throw new automCodeGenException(
                            automCodeGenException.ParserEOFExceptionCode);
                    }
                }
            } // end of While

        }catch (IOException e){
            throw new automCodeGenException(
                automCodeGenException.ParserExceptionCode);
        }
    }

    /**
     * getNextToken, check if the next token from the input stream is
     * the same as the input reserve word.
     * @param waitOn, the reserve word to be used for compare
     * @return void,
     * @exception automCodeException, IOException.
     */
    public void getNextToken(int waitOn) throws automCodeGenException,
    IOException{
        String tempStr;
        if (waitOn != EOL){
            while (true){
                inStreamTokenizer.nextToken();
                if (inStreamTokenizer.ttype != StreamTokenizer.TT_EOF){
                    if (inStreamTokenizer.ttype == StreamTokenizer.TT_WORD){
                        tempStr = new String(inStreamTokenizer.sval);
                        if (tempStr.compareTo(reserveWords[waitOn])==0){
                            break;
                        }else {
                            // not the correct waitOn type

```



```

}

/**
 * Method to define the main production of the grammars.
 */
public boolean mainProduction() throws automCodeGenException{
    int currentProduction = 0 ;
    String tempS = "";
    String tempS2;
    String tempS3;

    try {
        while (true){
            switch (currentProduction){
                //Type
                case 0 : getNextToken(TYPE);
                        currentProduction = 10;
                        break;
                case 10: tempS = new String (getNextToken(false));
                        // initialise a new class definition without any parameters
                        mainServerClass = new classDefinition();
                        mainServerClass.setName(tempS);
                        currentProduction = 20;
                        break;
                //Specification
                case 20: getNextToken(SPECIFICATION);
                        currentProduction = 30;
                        break;

                // get the identification variables
                case 30: tempS = new String (getNextToken(false));
                        if(tempS.compareTo(reserveWords[END]) == 0){
                            currentProduction = 40;
                        }else {
                            tempS2 = new String(getNextToken(false));
                            mainServerClass.addID(tempS,tempS2);
                        }
                        break;

                // OPERATOR & IMPLEMENTATION
                case 40: tempS = new String(getNextToken(false));
                        if (tempS.compareTo(reserveWords[OPERATOR]) == 0){
                            operatorProduction();
                        } else if (tempS.compareTo(
                                reserveWords[IMPLEMENTATION]) == 0){
                            currentProduction = 50;
                        } else {
                            throw new automCodeGenException(
                                automCodeGenException.ParserSyntaxError);
                        }
                        break;

                case 50: tempS = new String(getNextToken(false));
                        if (tempS.compareTo(reserveWords[SPACE]) == 0){
                            spaceProduction();

```

```

        currentProduction = 60;
    }else if (tempS.compareTo(reserveWords[END]) == 0){
        currentProduction = 60;
    }else if (tempS.compareTo(reserveWords[JAVA]) == 0){
        mainServerClass.setObjectState(false);
        tempS2 = new String(getNextToken(false));
        mainServerClass.setPackageName(tempS2);
    }
    break;
case 60: distObject.addElement(mainServerClass);
tempS = new String(getNextToken(false));
if (tempS.compareTo(reserveWords[TYPE]) == 0){
    currentProduction = 10;
}else {
    throw new automCodeGenException(
        automCodeGenException.ParserSyntaxError);
}
break; // end state, when the EOF is reach

} // end of switch
} // end of while
} catch (automCodeGenException e) {
    if ((e.errorCode ==
automCodeGenException.ParserEOFExceptionCode)
        & (currentProduction == 60)) {
        System.err.println("Phase 1 : Parser Completed without
error");
        return (true);
    }
    System.err.println(" *** Parser error detected ***");
    e.printStackTrace();
    System.err.println(" *** On line number " +
        inStreamTokenizer.lineno());
    return (false);
} catch (IOException e){
    System.err.println(" *** IO error detected ***");
    System.err.println(" *** On line number " +
        inStreamTokenizer.lineno());
    return (false);
}
} // end of main production

/**
 * operatorProduction, parse the methods of the objects
 * @exception, automCodeGenException and IOException
 * @return void
 */
private void operatorProduction() throws automCodeGenException,
IOException{
    int currentProduction = 0;
    String stemp1;
    String stemp2;
    methodDefinition currentMethod = new methodDefinition("");

    while (true){
        switch (currentProduction){

```

```

case 0 : stemp1 = new String (getNextToken(false));
        currentMethod = mainServerClass.addMethod(stemp1);
        currentProduction = 10;
        break;
case 10: getNextToken(SPECIFICATION);
        currentProduction =20;
        break;

case 20: stemp1 = new String (getNextToken(false));
        if (stemp1.compareTo(reserveWords[INPUT]) == 0){
            stemp1 = new String (getNextToken(false));
            stemp2 = new String (getNextToken(false));
            currentMethod.addParameter
                (definition.IN,stemp1,stemp2);
        }else if (stemp1.compareTo(reserveWords[OUTPUT]) ==
0){

            // the first output type will be the method return
            // type
            stemp1 = new String (getNextToken(false));
            stemp2 = new String (getNextToken(false));
            currentMethod.setRetType(stemp2);
            currentProduction =30;
        } else if (stemp1.compareTo(reserveWords[STATES])
== 0){
            // the state type must not be of primitive type
            stemp1 = new String (getNextToken(false));
            stemp2 = new String (getNextToken(false));
            if (variableDefinition.javaType(stemp2) !=
                definition.SUSERTYPE){
                throw new automCodeGenException(
                    automCodeGenException.ParserStateError);
            }
        }

        currentMethod.addParameter(definition.INOUT,stemp1,
                                stemp2);
        }else if (stemp1.compareTo(reserveWords[END]) == 0){
            currentProduction =40;
        }
        break;
case 30: stemp1 = new String (getNextToken(false));
        if (stemp1.compareTo(reserveWords[INPUT]) == 0){
            stemp1 = new String (getNextToken(false));
            stemp2 = new String (getNextToken(false));
            currentMethod.addParameter(definition.IN,stemp1,
                                stemp2);
        }else if (stemp1.compareTo(reserveWords[OUTPUT]) ==
0){

            // the first output type will be the method return
            // type
            stemp1 = new String (getNextToken(false));
            stemp2 = new String (getNextToken(false));
            if (variableDefinition.javaType(stemp2) !=
                definition.SUSERTYPE){
                throw new automCodeGenException(
                    automCodeGenException.ParserStateError);
            }
        }

```

```

currentMethod.addParameter(definition.INOUT, stemp1,
                           stemp2);
    } else if (stemp1.compareTo(reserveWords[STATES])
               == 0){
        // the state type must not be of primitive type
        stemp1 = new String (getNextToken(false));
        stemp2 = new String (getNextToken(false));
        if (variableDefinition.javaType(stemp2) !=
            definition.SUSERTYPE){
            throw new automCodeGenException(
                automCodeGenException.ParserStateError);
        }
    }

currentMethod.addParameter(definition.INOUT, stemp1,
                           stemp2);
    }else if (stemp1.compareTo(reserveWords[END]) == 0){
        currentProduction = 40;
    }
    break;

case 40: getNextToken(IMPLEMENTATION);
        currentProduction = 50;
        break;
case 50: stemp1 = new String(getNextToken(false));
        if (stemp1.compareTo(reserveWords[SPACE]) == 0){
            spaceMethodProduction(currentMethod);
        }else if (stemp1.compareTo(reserveWords[END]) == 0){
            return;
        }else {
            throw new automCodeGenException(
                automCodeGenException.ParserSyntaxError);
        }
        break;
    } // end of switch
} // end of while
} // end of operatorProduction

private void spaceMethodProduction(methodDefinition currentMethod)
throws automCodeGenException, IOException{
    int currentProduction = 0;
    String stemp1;
    String stemp2;

    while (true){
        switch (currentProduction){
            case 0: getNextToken(PROPERTY);
                    currentProduction = 10;
                    break;
            case 10: stemp1 = new String(getNextToken(false));
                     if (stemp1.compareTo(reserveWords[SPACEMODE]) == 0){
                         // space mode
                         stemp2 = new String(getNextToken(false));
                     }

currentMethod.setReadProperty(definition.spaceTranslate(stemp2));
        currentProduction = 20;
        }else if (stemp1.compareTo(reserveWords[TRANSACTION])

```

```

        == 0){
            // trasaction mode Yes, No
            stemp2 = new String(getNextToken(false));

currentMethod.setTransaction(definition.spaceTranslate(stemp2));
            currentProduction = 20;
        }else if
(stemp1.compareTo(reserveWords[TRANSACTIONTIME])
        == 0){
            // trasaction time in ms
            long itemp = getNumNextToken(false);
            currentMethod.setTransactionTime(itemp);
            currentProduction = 20;
        }else {
            throw new automCodeGenException(
                automCodeGenException.ParserSyntaxError);
        }

        break;

        case 20: stemp1 = new String(getNextToken(false));
            if (stemp1.compareTo(reserveWords[PROPERTY]) == 0){
                currentProduction = 10;
            }else if (stemp1.compareTo(reserveWords[END]) == 0){
                // End of IMPLEMENTATION
                return;
            }else {
                throw new automCodeGenException(
                    automCodeGenException.ParserSyntaxError);
            }
        }
    } // end of while
} // end of method
private void spaceProduction() throws automCodeGenException,
IOException{
    int currentProduction = 0;
    String stemp1;
    String stemp2;
    spaceProperties objectSP;

    objectSP = mainServerClass.getProperties();

    while (true){
        switch (currentProduction){
            case 0: getNextToken(PROPERTY);
                currentProduction = 10;
                break;
            case 10: stemp1 = new String(getNextToken(false));
                if (stemp1.compareTo(reserveWords[SPACENAME]) == 0){
                    // space name
                    objectSP.setName(getNextToken(false));
                    currentProduction = 20;
                }else if (stemp1.compareTo(reserveWords[OWNERSHIP])
                    == 0){
                    // ownership mode
                    objectSP.setOwnership(definition.spaceTranslate

```

```

        (getNextToken(false)));
        currentProduction = 20;
    }else if (stemp1.compareTo(reserveWords[SECURITY]) ==
0){
        // security mode
        objectSP.setSecurity(definition.spaceTranslate
        (getNextToken(false)));
        currentProduction = 20;
    } else if (stemp1.compareTo(reserveWords[LEASE]) ==
0){
        // lease mode in ms
        long itemp = (long) getNumNextToken(false);
        objectSP.setLease(itemp);
        currentProduction = 20;
    }else if (stemp1.compareTo(reserveWords[CLONE]) == 0){
        // clone mode
        objectSP.setClone(definition.spaceTranslate
        (getNextToken(false)));
        currentProduction = 20;
    }else if (stemp1.compareTo(reserveWords[NOTIFY]) ==
0){
        // notify mode
        objectSP.setNotify(definition.spaceTranslate
        (getNextToken(false)));
        currentProduction = 20;
    }else {
        throw new automCodeGenException(
            automCodeGenException.ParserSyntaxError);
    }
    case 20: stemp1 = new String(getNextToken(false));
    if (stemp1.compareTo(reserveWords[PROPERTY]) == 0){
        currentProduction = 10;
    }else if (stemp1.compareTo(reserveWords[END]) == 0){
        // End of IMPLEMENTATION
        return;
    }else {
        throw new automCodeGenException(
            automCodeGenException.ParserSyntaxError);
    }
}
} // end of while
} // end of method

public Vector getDistObject(){
    return distObject;
}

} // end of class

```

APPBUILDER.JAVA

```

//-----
// Filename : appBuilder.java

```

```

// Date      : 10 Jan 00
// Compiler  : jdk 1.2 or J++
//-----

package psdl2java.generator;

/**
 *
 * @author Ngom Cheng
 * @version 1.0
 * appBuilder, generates all the interface wrapper
 * codes.
 */

import java.util.Vector;
import java.util.Enumeration;
import java.io.*;
import java.util.StringTokenizer;

// psdl2java classes
import psdl2java.global.*;
import psdl2java.parser.*;

public class appBuilder extends Object {

    /**
     * distObj, list of objects
     */
    private Vector distObj;

    /**
     * aicgToken[], array of reserve words
     */
    private String aicgToken[];

    /**
     * aicgField[], array of in String words
     */
    private String aicgField[];

    /**
     * sP, space properties
     */
    private spaceProperties sP;

    /**
     * cDef, main class definition
     */
    private classDefinition cDef;

    /**
     * destDir, destination directory
     */
    private String destDir;

    /**
     * dataDir, data directory

```

```

    */
    private String dataDir;

    /**
     * checkAgain, retry boolean
     */
    private boolean checkAgain = false;
    /**
     * static constants of the tokens
     */
    private static final int USEROBJECT = 0;
    private static final int USEREXT = 1;
    private static final int USERENTRY = 2;
    private static final int USERCLIENT = 3;
    private static final int USERSERVER = 4;
    private static final int USEROBJID = 5;
    private static final int USEROBJIDGETS = 6;
    private static final int DATE = 7;
    private static final int LOCALOBJNAME = 8;
    private static final int SPACENAME = 9;
    private static final int PACKAGENAME = 10;
    private static final int LOCALSPACENAME = 11;

    private static final int USERMETHOD = 12;
    private static final int USERDECLARATION = 13;
    private static final int USERPARAMETER = 14;

    private static final int ENTRYDECLARATION = 15;
    private static final int ENTRYASSIGN = 16;
    private static final int EXTDECLARATION = 17;
    private static final int AUTOGETID = 18;
    private static final int AUTOSETID = 19;
    private static final int EXTIDNULL = 20;
    private static final int CLIENTCONSTRUCTOR = 21;
    private static final int CLIENTCONSTRUCTASS1 = 22;
    private static final int CLIENTCONSTRUCTASS2 = 23;
    private static final int CLIENTMETHOD = 24;
    private static final int SERVERCONSTRUCTOR = 25;
    private static final int SERVERMETHOD = 26;
    private static final int SETGETID = 27;
    private static final int USERPARAMETERWC = 28;
    private static final int EVENTGETID = 29;
    private static final int DISTOBJGETID = 30;

    private static final char DELIMITER = '%';

    /**
     * Constructor Creates new appBuilder
     */
    public appBuilder(psdReader pReader, String destDir, String
dataDir) {
        //
        this.destDir = destDir;
        this.dataDir = dataDir;
        definition.DEFAULTPACKAGENAME = this.destDir;
        // initialise the tokenfield
        aicgToken = new String[definition.MAXTOKEN];

```

```

aicgField = new String[definition.MAXTOKEN];
aicgToken[USEROBJECT] = new String("-userobject");
aicgToken[USEREXT] = new String("-userext");
aicgToken[USERENTRY] = new String("-userentry");
aicgToken[USERCLIENT] = new String("-userclient");
aicgToken[USERSERVER] = new String("-userserver");
aicgToken[USEROBJID] = new String("-userobjid");
aicgToken[USEROBJIDGETS] = new String("-userobjidgets");
aicgToken[DATE] = new String("-date");
aicgToken[LOCALOBJNAME] = new String("-localobjname");
aicgToken[SPACENAME] = new String("-spacename");
aicgToken[PACKAGENAME] = new String("-packagename");
aicgToken[LOCALSPACENAME] = new String("-localspacename");

//
aicgToken[USERDECLARATION] = new String("-userdeclaration");
aicgToken[USERMETHOD] = new String("-usermethod");
aicgToken[ENTRYDECLARATION] = new String("-entrydeclaration");
aicgToken[EXTDECLARATION] = new String("-extdeclaration");
aicgToken[USERPARAMETER] = new String("-userparameter");
aicgToken[USERPARAMETERWC] = new String("-userparameterwc");
aicgToken[USERDECLARATION] = new String("-userdeclaration");
aicgToken[ENTRYASSIGN] = new String("-entryassign");
aicgToken[AUTOGETID] = new String("-autogetid");
aicgToken[AUTOSETID] = new String("-autosetid");
aicgToken[EXTIDNULL] = new String("-extidnull");
aicgToken[CLIENTCONSTRUCTOR] = new String("-clientconstructor");
aicgToken[CLIENTCONSTRUCTASS1] = new String("-clientconstructass1");
aicgToken[CLIENTCONSTRUCTASS2] = new String("-clientconstructass2");
aicgToken[CLIENTMETHOD] = new String("-clientmethod");
aicgToken[SERVERMETHOD] = new String("-servermethod");
aicgToken[SERVERCONSTRUCTOR] = new String("-serverconstructor");
aicgToken[SETGETID] = new String("-setgetid");
aicgToken[EVENTGETID] = new String("-eventgetid");
aicgToken[DISTOBJGETID] = new String("-distobjgetid");

//
aicgField[LOCALOBJNAME] = new String("inObj");
aicgField[PACKAGENAME] = new
String(definition.DEFAULTPACKAGENAME);
aicgField[DATE] = new String("01/07/00");
aicgField[SPACENAME] = new String(definition.DEFAULTSPACENAME);

// create the package
createPackage();
// create the fixed classes
buildFixedClasses();
// get the distributed objects from parser
distObj = pReader.getDistObject();
// get a list out
Enumeration enum = distObj.elements();
while (enum.hasMoreElements()){
    // get the distributed class object
    cDef = (classDefinition) enum.nextElement();
    if (cDef.getObjectState()){

```

```

        // it a aicg object
        // build the name
        aicgField[USEROBJECT] = new String(cDef.getName());
        aicgField[USEREXT] = new String(aicgField[0] + "Ext");
        aicgField[USERENTRY] = new String(aicgField[0] + "Entry");
        aicgField[USERCLIENT] = new String(aicgField[1] + "Client");
        aicgField[USERSERVER] = new String(aicgField[1] + "Server");

        // get the space properties
        sP = cDef.getProperties();
        aicgField[LOCALSPACENAME] = sP.getName();
        // process the information
        processDistObj();
        // build the distributed object
        buildDistObj();
    }

    } // end of while

}

/**
 * createPackage, create a directory for the interface wrapper
 * @return void
 */
public void createPackage(){

    // create the files for that application
    File name = new File(aicgField[PACKAGENAME]);
    System.out.println("Phase 2 : Generating...");
    if (name.exists()){
        System.out.println("          : OutputDirectory Exists");
        System.out.println("          : No new directory created ");
    }else {
        try{
            name.mkdir();
            System.out.println("          : " + name.getName() + "
created");
        }catch (Exception e){
            System.err.println("          : Cannot create directory...
            exiting");
            System.exit(-1);
        }
    }
}

/**
 * buildClass, convert the template data into interface wrapper
 * @param inName, name of the template
 * @param outName, name of the output file
 * @return void
 */
public void buildClass(String inName, String outName){
    PrintWriter output;
    BufferedReader input;
    String sTemp, sTemp2;

```

```

StringTokenizer sTokenizer;
// open the file entryAICG.java.data
try{
    input = new BufferedReader(new FileReader(inName));
    output = new PrintWriter(new BufferedWriter(new
        FileWriter(aicgField[PACKAGENAME] + outName)));
    try{
        // copy the input to a temporary String

        while ((sTemp = input.readLine()) != null){
            sTemp = convert(sTemp);
            while(checkAgain){
                checkAgain = false;
                sTemp = convert(sTemp);
            }
            output.println(sTemp);
        }
    }catch (IOException e1){
        System.err.println("Error in creating " + inName);
        System.exit(1);
    }
    output.close();
    input.close();
}catch (IOException e){
    System.err.println("Error in creating " + inName);
    System.exit(1);
}
}

/**
 * buildFixedClass, build all the AICG fixed classes
 * @return void
 */
public void buildFixedClasses(){
    // open the file globalAICG.java.data
    buildClass( dataDir + "\\globalAICG.java.data",
        "\\globalAICG.java");
    // open the file entryAICG.java.data
    buildClass(dataDir + "\\entryAICG.java.data",
        "\\entryAICG.java");
    // open the file exceptionAICG.java.data
    buildClass(dataDir + "\\exceptionAICG.java.data" ,
        "\\exceptionAICG.java");
    // open the file eventAICGID.java.data
    buildClass(dataDir + "\\eventAICGID.java.data",
        "\\eventAICGID.java");
    // open the file notifyAICGID.java.data
    buildClass(dataDir + "\\notifyAICG.java.data",
        "\\notifyAICG.java");
    // open the file spaceAICGID.java.data
    buildClass(dataDir + "\\spaceAICG.java.data",
        "\\spaceAICG.java");
    // open the file transactionAICG.java.data
    buildClass(dataDir + "\\transactionAICG.java.data",
        "\\transactionAICG.java");
    // open the file spaceAICGID.java.data
    buildClass(dataDir + "\\transactionManagerAICG.java.data",

```

```

        "\\transactionManagerAICG.java");
    }

    /**
     * cleasAICGField, reset the AICG Fields
     * @return void
     */
    public void clearAICGField(){
        aicgField[USERDECLARATION] = new String();
        aicgField[USEROBJID] = new String();
        aicgField[ENTRYDECLARATION] = new String();
        aicgField[EXTDECLARATION] = new String();
        aicgField[USERPARAMETER] = new String();
        aicgField[ENTRYASSIGN] = new String();
        aicgField[AUTOGETID] = new String();
        aicgField[AUTOSETID] = new String();
        aicgField[SETGETID] = new String();
        aicgField[EXTIDNULL] = new String();
        aicgField[CLIENTCONSTRUCTASS1] = new String();
        aicgField[CLIENTCONSTRUCTASS2] = new String();
        aicgField[USERPARAMETERWC] = new String();
        aicgField[EVENTGETID] = new String();
    }

    /**
     * processDistObj, gather information of the distributed object
     * for use in creating the interface
     * @return void
     */
    public void processDistObj(){
        variableDefinition vD;
        methodDefinition mD;
        Enumeration enum;
        Vector iDs;
        int counter = 0;
        // clear the aicgField to empty
        clearAICGField();
        iDs = cDef.getID();
        enum = iDs.elements();
        while(enum.hasMoreElements()){
            vD = (variableDefinition) enum.nextElement();
            if (counter == 0){
                aicgField[USERDECLARATION] = new String("private " +
                    vD.getJavaObject() + " " + vD.getID() + ";\n");
                aicgField[USEROBJID] = new
String(vD.getClassID().getAICGName()
                    + ", ");
                aicgField[ENTRYDECLARATION] = new String("public " +
                    vD.getJavaObject() +
                    " " + vD.getClassID().getAICGName() + ";\n");
                aicgField[EXTDECLARATION] = new String("protected " +
                    vD.getJavaObject() + " " +
                    vD.getClassID().getAICGName() + ";\n");
                aicgField[USERPARAMETER] = new String(vD.getJavaObject() +
                    " " + vD.getClassID().getAICGName());
                aicgField[USERPARAMETERWC] = new String(vD.getJavaObject() +
                    " " + vD.getClassID().getAICGName() + ", ");
            }
            counter++;
        }
    }

```

```

aicgField[ENTRYASSIGN] = new String("this." +
    vD.getClassID().getAICGName() + " = " +
    vD.getClassID().getAICGName() + ";\n");
aicgField[AUTOGETID] = new String("public synchronized "
    +vD.getJavaObject() + ' ' + "get" +
    vD.getClassID().getAICGName() + "()\n" +
    " {\n    return " + vD.getClassID().getAICGName()
+
    ";\n } \n");
aicgField[DISTOBJGETID] = new String("public synchronized "
    +vD.getJavaObject() + ' ' + "get" +
    vD.getClassID().getAICGName() + "()\n" +
    " {\n    return " + vD.getID() + ";\n } \n");
+
aicgField[AUTOSETID] = new String("public synchronized void "
    "set" + vD.getClassID().getAICGName() + "( " +
    vD.getJavaObject() + " inID)\n" + " {\n    " +
    vD.getClassID().getAICGName() + " = inID;\n } \n");
aicgField[SETGETID] = new String("    set" +
    vD.getClassID().getAICGName() + "(ddata.get" +
    vD.getClassID().getAICGName() + "());\n");
aicgField[EXTIDNULL] = new
String(vD.getClassID().getAICGName()
    + " = null;\n");
aicgField[CLIENTCONSTRUCTASS1] = new String("    " +
    vD.getJavaObject() + ' ' + "temp" +
    vD.getClassID().getAICGName() + " = " +
    vD.getClassID().getAICGName() + " ;\n");
aicgField[CLIENTCONSTRUCTASS2] = new String(", temp" +
    vD.getClassID().getAICGName());
aicgField[EVENTGETID] = new String("indata.get" +
    vD.getClassID().getAICGName() + "(), ");
} else {
    aicgField[USERDECLARATION] = new
        String(aicgField[USERDECLARATION] +
            " private " + vD.getJavaObject() + " " +
vD.getID()
            + ";\n");
    aicgField[USEROBJID] = new String(aicgField[USEROBJID]
        + vD.getClassID().getAICGName() + ", ");
    aicgField[ENTRYDECLARATION] = new
        String(aicgField[ENTRYDECLARATION] +
            " public " + vD.getJavaObject() +
            " " + vD.getClassID().getAICGName() + ";\n");
    aicgField[EXTDECLARATION] = new
String(aicgField[EXTDECLARATION]
        + " protected " + vD.getJavaObject() +
        " " + vD.getClassID().getAICGName() + ";\n");
    aicgField[USERPARAMETER] = new
String(aicgField[USERPARAMETER] +
        ", " + vD.getJavaObject() +
        " " + vD.getClassID().getAICGName());
    aicgField[USERPARAMETERWC] = new
        String(aicgField[USERPARAMETERWC]
            + vD.getJavaObject() +
            " " + vD.getClassID().getAICGName() + ", ");

```

```

aicgField[ENTRYASSIGN] = new String(aicgField[ENTRYASSIGN] +
    "    this." + vD.getClassID().getAICGName() +
    " = " + vD.getClassID().getAICGName() + ";\n");
aicgField[AUTOGETID] = new String(aicgField[AUTOGETID] +
    "    public synchronized " + vD.getJavaObject() + ' ' +
    "get" + vD.getClassID().getAICGName() + "()\n" +
    "    {\n        return " + vD.getClassID().getAICGName() +
    ";\n    }\n");
aicgField[DISTOBJGETID] = new String(aicgField[DISTOBJGETID] +
    "    public synchronized " + vD.getJavaObject() + ' ' +
    "get" + vD.getClassID().getAICGName() + "()\n" +
    "    {\n        return " + vD.getID() + ";\n    }\n");
aicgField[AUTOSETID] = new String(aicgField[AUTOSETID] +
    "    public synchronized void " + "set" +
    vD.getClassID().getAICGName() + "( " +
    vD.getJavaObject() + " inID)\n" +
    "    {\n        " + vD.getClassID().getAICGName() + " =
    inID;\n    }\n");
aicgField[SETGETID] = new String(aicgField[SETGETID] +
    "    set" + vD.getClassID().getAICGName() +
    "(ddata.get" + vD.getClassID().getAICGName() +
    "());\n");
aicgField[EXTIDNULL] = new String(aicgField[EXTIDNULL] +
    "    " + vD.getClassID().getAICGName() + " =
    null;\n");
aicgField[CLIENTCONSTRUCTASS1] = new String(
    aicgField[CLIENTCONSTRUCTASS1] + "    " +
    vD.getJavaObject() + ' ' + "temp" +
    vD.getClassID().getAICGName() + " = " +
    vD.getClassID().getAICGName() + " ;\n");
aicgField[CLIENTCONSTRUCTASS2] = new
    String(aicgField[CLIENTCONSTRUCTASS2] +
    ", temp" + vD.getClassID().getAICGName());
aicgField[EVENTGETID] = new String(aicgField[EVENTGETID] +
    "indata.get" + vD.getClassID().getAICGName() + "()",
    ");
}
counter ++;
} //end of while

// process the methods
iDs = cDef.getMethods();
enum= iDs.elements();
aicgField[CLIENTMETHOD] = new String();
aicgField[SERVERMETHOD] = new String();
String withcomma;
counter = 0;
while(enum.hasMoreElements()){
    mD = (methodDefinition) enum.nextElement();
    // constructor for client
    if (mD.getReadProperty() == definition.SPCONSTRUCTOR){
        aicgField[CLIENTCONSTRUCTOR] = new String("public " +
            mD.getRetType() + " " + aicgField[USERCLIENT] + "( "
            + aicgField[USERPARAMETER] +
            ") throws exceptionAICG{\n" + "    super();\n" +

```

```

"    int retry = globalAICG.AICGDEFAULTRETRY;\n" +
"    boolean trxSucc = false;\n" +
"aicgField[CLIENTCONSTRUCTASS1] +
"        while (retry != 0){\n" + "            try{\n" +
"                // create atemplate to search for the
object\n"
+ "            " + aicgField[USERENTRY] +
" template = new " + aicgField[USERENTRY] +
" (globalAICG.AICGIDENTIFIER " +
" aicgField[CLIENTCONSTRUCTASS2] +
", null);\n" +
"            distObj = (" + aicgField[USERENTRY] +
") space.read(template,null,\n" +
"                globalAICG.AICGDEFAULTWAITTIME);\n" +
"            " + aicgField[USEROBJECT] + " ddata = (" +
aicgField[USEROBJECT] +
") distObj.getObject();\n" +
"            " + aicgField[SETGETID] +
"            retry = 0;\n" +
"            trxSucc = true;\n" +
"            }catch (Exception e){\n" +
"                System.err.println(\"retrying to find
object\");\n" +
"                retry--;\n" +
"            }\n" +
"        } // end of while\n" +
"        if (!trxSucc){\n" +
"            throw new exceptionAICG
(exceptionAICG.ObjectNotFoundException);\n" +
"        } \n" +
"    }\n" );

//server constructor
aicgField[SERVERCONSTRUCTOR] = new String("public " +
mD.getRetType() + " " +
aicgField[USERSERVER] +
"( " + mD.getParameterString() +
") throws exceptionAICG{\n" +
"    super();\n" +
"    // create the template\n" +
"    Transaction trn = null;\n" +
"    try {\n" +
"        trn = trnMgr.getDefaultTransaction();\n" +
"        try{\n" +
"            " + aicgField[USEROBJECT] + " ddata = new
" +
" aicgField[USEROBJECT] + '(' +
mD.getParameterName() + ");\n" +
"            " + aicgField[SETGETID] +
"            " + aicgField[USERENTRY] + " template = new
"+
"            aicgField[USERENTRY] +
"(globalAICG.AICGIDENTIFIER, " +
" aicgField[USEROBJID] + "null);\n"
+
"            //check if the distobj exit\n" +
"            if (space.read(template, trn,

```

```

        JavaSpace.NO_WAIT) == null){\n" +
        "    distObj = new " + aicgField[USERENTRY] +
        "(globalAICG.AICGIDENTIFIER, " +
        aicgField[USEROBJID] + "ddata);\n" +
        "    ,    space.write(distObj, trn,
        globalAICG.AICGLEASETIME);\n" +
        "    trn.commit();\n" +
        "    }else{ \n" +
        "        // throws an exception\n" +
        "        trn.abort();\n" +
        "        throw new
        exceptionAICG(
        exceptionAICG.ObjectExistException);\n" +
        "    }\n" +
        "    }catch (exceptionAICG e){\n" +
        "        trn.abort();\n" +
        "        throw new
exceptionAICG(e.getErrorCode());\n"+
        "    }catch (Exception e) {\n" +
        "        trn.abort();\n" +
        "        e.printStackTrace();\n" +
        "        throw new exceptionAICG(
        exceptionAICG.SystemExceptionCode);\n" +
        "    }\n" +
        "    }catch (exceptionAICG e) {\n" +
        "        throw new exceptionAICG(e.getErrorCode());\n"
+
        "    }catch (Exception e) {\n" +
        "        throw new exceptionAICG(
        exceptionAICG.SystemExceptionCode);\n" +
        "    }\n" +
        " } \n");

}else if (mD.getReadProperty() == definition.SPREAD){
    aicgField[CLIENTMETHOD] = new String(aicgField[CLIENTMETHOD]
+
        "    public " + mD.getRetType() + "    " +
        mD.getName() + "( " +
        mD.getParameterString() + ") throws
        exceptionAICG {\n" +
        "        " + aicgField[USERENTRY] + "
        msgTemplate = new " +
        aicgField[USERENTRY] +
        "(globalAICG.AICGIDENTIFIER, " +
        aicgField[USEROBJID] + "null);\n\n" +
        "        try{\n" +
        "            distObj = (" +
        aicgField[USERENTRY] + ")\n" +
        "            space.read(msgTemplate,
null,
        globalAICG.AICGDEFAULTWAITTIME);\n" +
        "            if (distObj==null){\n" +
        "                // object no longer exist \n"
+
        "                throw new exceptionAICG(

```

```

exceptionAICG.ObjectNotFoundException);\n"
+ "    }\n" +
    "    return distObj.data." +
mD.getName() + '(' + mD.getParameterName()
+ ");\n" +
    "    }catch (exceptionAICG ex){\n" +
    "        throw new exceptionAICG(
exceptionAICG.ObjectNotFoundException);\n"
+ "    }catch (Exception e) {\n" +
    "        throw new exceptionAICG(
exceptionAICG.SystemExceptionCode);\n" +
    "    }\n" +
    "    }\n\n");
}else {
    aicgField[SERVERMETHOD] = new String(aicgField[SERVERMETHOD]
+
    "    public " + mD.getRetType() + " " +
    mD.getName() + "( " +
    mD.getParameterString() + ") throws
exceptionAICG {\n" +
    "        int retry =
globalAICG.AICGDEFAULTRETRY;\n" +
    "        boolean trxSucc =false;\n" +
    "        " + aicgField[USERENTRY] + "
msgTemplate =\n" +
    "        new " + aicgField[USERENTRY]
+
    "(globalAICG.AICGIDENTIFIER, " +
    aicgField[USEROBJID] + "null);\n" +
    "        Transaction trn = null;\n\n" +
    "        while(retry !=0){\n" +
    "            try {\n" +
    "                trn = trnMgr.createTransaction
(globalAICG.USERTXNLEASE);\n" +
    "                try{\n" +
    "                    distObj = (" +
    aicgField[USERENTRY] + ")\n" +
    "                    space.take(msgTemplate,
trn,
    Long.MAX_VALUE);\n" +
    "                    // called the actual
method\n" +
    "                    distObj.data." +
mD.getName() +
    '(' + mD.getParameterName() + ");\n" +
    "                    space.write (distObj,
trn,globalAICG.AICGLEASETIME);\n" +
    "                    trn.commit();\n" +
    "                    retry = 0;\n" +
    "                    trxSucc = true;\n" +
    "                }catch (Exception e){\n" +
    "                    trn.abort();\n" +
    "                    retry --;\n" +
    "                }\n\n" +
    "            }\n\n" +
    "        }\n\n" +
    "        System.err.println(\"Exception
level 1: retrying\");\n" +

```

```

        } \n" +
        "    } catch (Exception e) { \n" +
        "        retry--; \n" +
        "        System.err.println(\"Exception\n" +
        "level 2: retrying\"); \n" +
        "    } \n" +
        "    } // end of while \n" +
        "    if (!trxSucc) { \n" +
        "        throw new exceptionAICG\n" +
        "        (exceptionAICG.SystemExceptionCode); \n" +
        "    } \n" + "    } \n" + "    } \n");
    }

    // others
    if (counter == 0) {
        aicgField[USERMETHOD] = new String("public " +
        mD.getRetType() +
        " " + mD.getName() + "(" +
        mD.getParameterString() + ")" +
        " { \n \n \n } \n \n");

    } else {
        aicgField[USERMETHOD] = new String(aicgField[USERMETHOD] +
        " public " + mD.getRetType() + " " +
        mD.getName() + "(" +
        mD.getParameterString() + ")" +
        " { \n \n \n } \n \n");
    }
    counter++;
}

}
/**
 * buildDistObj, build the rest of the interface wrappers
 * @return void
 */
public void buildDistObj() {
    PrintWriter output;
    BufferedReader input;
    String sTemp, sTemp2;
    StringTokenizer sTokenizer;
    // build the distributed object
    buildClass(dataDir + "\\distobj.java.data",
        "\\\" + aicgField[USEROBJECT] + ".java");
    // build the Entry object
    buildClass(dataDir + "\\distobjentry.java.data",
        "\\\" + aicgField[USERENTRY] + ".java");
    // build the Ext object
    buildClass(dataDir + "\\distobjext.java.data",
        "\\\" + aicgField[USEREXT] + ".java");
    // build the ExtClient object
    buildClass(dataDir + "\\distobjextclient.java.data",
        "\\\" + aicgField[USERCLIENT] + ".java");
    // build the ExtServer object
    buildClass(dataDir + "\\distobjextserver.java.data",

```

```

        "\\\" + aicgField[USERSERVER] + ".java");

    // build the eventAICGHandler object
    buildClass(dataDir + "\\eventAICGHandler.java.data",
        "\\eventAICGHandler.java");
    System.out.println("        : Interface Generating Completed");
}

/**
 * convert, parse the template statement into codes
 * @param inString, template statement
 * @return String, output statement
 */
public String convert(String inString){

    int len = 0;
    int endLen = 0;
    char cTemp;
    String sTemp;

    String retString = new String (inString);

    try{

        while (len < retString.length()){
            if (retString.charAt(len) == DELIMITER){
                if (inString.charAt(len + 1) == '-') {
                    len = len + 1;
                    endLen = len + 1;
                    while( retString.charAt(endLen) != DELIMITER){
                        endLen ++;
                    }
                    sTemp = new String( retString.substring(len,endLen));
                    sTemp = replace(sTemp);
                    if (endLen < retString.length()-1){
                        retString = new String(retString.substring(0, len-1) +
                            sTemp + retString.substring(endLen + 1,
                                retString.length()));
                    }else {
                        retString =new String(retString.substring(0, len-1)
                            + sTemp );
                    }
                    checkAgain = true;
                    return retString;
                }
            }
            len++;
        }
    }catch (Exception e){
        System.err.println("out of Bound error");
        System.exit(1);
    }
    return retString;
}

/**

```

```

* replace, check and replace the token with the respective string
* @param inString, token
* @return String, output codes for the token
*/
public String replace(String inString){
    for (int i = 0; i< definition.MAXFIX; i++){
        if (inString.compareTo(aicgToken[i]) ==0){
            return aicgField[i];
        }
    }

    System.err.println("Token not found " + inString );
    System.exit(-1);
    return inString;
}
}

```

THIS PAGE IS INTENTIONALLY LEFT BLANK

APPENDIX D. SETTING UP THE JAVASPACE ENVIRONMENT

EXTRACTED FROM THE NUTS AND BOLTS OF COMPILING AND RUNNING JAVASPACE PROGRAMS BY SUSANNE HUPFER

Downloading the Software Packages

Compiling and running JavaSpaces programs requires the installation of three separate packages from Sun Microsystems--Java, Jini, and JavaSpaces software.

- If you don't already have the Java™ 2 SDK, Standard Edition, v 1.2.x installed on your machine, download it from Sun's Java 2 SDK
<<http://www.java.sun.com/products/jdk/1.2/>>. The Jini and JavaSpaces software require version 1.2.x of the Java Development Kit, because they make use of RMI features that are not available in version 1.1 of the JDK.
- Next you'll need to download the Jini™ Technology Starter Kit, Version 1.0.1, from the Jini System Software 1.0 Product Offerings
<<http://developer.java.sun.com/developer/products/jini/product.offerings.html>>. (Note that, as this article is being written, Version 1.0.1 is the latest release, and you'll therefore see the number "1.0.1" appearing in the commands; if there is a newer release when you read this article, you should substitute the newer version numbers as appropriate.) You'll have to click on the link "Register and accept the SCSL to access software downloads" and step through a free registration process in order to download the kit. The file you download is called `jini1_0_1.zip`. The Jini Technology Starter Kit contains Jini interfaces and classes, along with a number of services. It also contains the complete set of Sun Microsystems Jini specifications, documentation, and example code.
- Last, download the JavaSpaces™ Technology Kit (JSTK), Version 1.0.1, from the Jini System Software 1.0 Product Offerings
<<http://developer.java.sun.com/developer/products/jini/product.offerings.html>>. Again, you'll need to click on the link "Register and accept the SCSL to access

software downloads." The file you download is called `jstkl_0_1.zip`; it contains the Sun Microsystems implementation of the JavaSpaces technology, along with documentation and example code.

Installing the Packages

- Now that you've downloaded the three necessary software packages, you're ready to unpack and install them. First, if you don't already have Java™ 2 SDK™ version 1.2.x installed, follow the installation instructions that come with the package to install it on your platform. This article assumes that Java software is installed in `C:\jdk1.2.x\` on Windows platforms and `/jdk1.2.x/` on Solaris platforms.
- Next, you need to install the Jini Technology Starter Kit and then the JavaSpaces Technology Kit (JSTK). Note that the current packaging of the JSTK requires that the Jini software be installed first, because the software packages build upon one another; if this order is not followed, the installation and API documentation will not be correct. Here are the steps in the correct order:
- Extract the files in `jini1_0_1.zip` to your chosen destination directory, using a ZIP extraction utility. For example, on the Windows platform, use a utility such as WinZip and specify that the software distribution should extract to a certain destination directory. If you specify `C:\` as the destination, you should now have a folder `C:\jini1_0_1` that contains the Jini files. On the Solaris platform, copy the file `jini1_0_1.zip` to a destination directory, `cd` to that directory, and then use a ZIP extraction program, for example `jar -xvf jini1_0_1.zip`. This document assumes that Jini software resides in `C:\jini1_0_1\` on Windows platforms and `/jini1_0_1/` on Solaris platforms.
- Extract the files in `jstkl_0_1.zip` to the same destination directory. This results in some files being overwritten, so when asked whether you wish to replace any of the existing files, answer "yes".

Configuring Your Machine Environment

Now that you have unpacked the software, you still need to do some configuration. The file `C:\jini1_0_1\index.html` (`//jini1_0_1/index.html` on Solaris) contains pointers to release notes and other documentation (much of which is found in the `doc` folder). You may want to read over the release notes for any information not covered in this document; in particular, the sections covering "Known Bugs" or "Known Issues" can be a useful starting point if you're experiencing a problem.

Make sure that the path to the basic tools provided by the Java Development Kit (for example, `java`, `javac`, and `appletviewer`) is specified in your executable path. For instance, on Windows platforms, you need to set the `PATH` environment variable as follows:

```
set PATH=%PATH%;C:\jdk1.2.x\bin;
```

As for your `CLASSPATH` environment variable, we recommend that you do not modify it. To run the standard run-time Jini services (see Starting the Required Run-time Services), no special classpath is needed. When you build and run your own JavaSpaces code, we recommend passing the required JAR files on the command line, rather than modifying the classpath or placing JAR files in the Java extensions directory (we'll explain further in Compiling JavaSpaces Programs).

Starting the Required Run-time Services

JavaSpaces applications interact with one or more spaces, so to run a JavaSpaces application you'll need to have at least one JavaSpaces service running. But this itself depends on having a Jini infrastructure in place, and before you start a JavaSpaces service, you need to start these other run-time services:

- An **HTTP server**, which is used to download code to JavaSpaces clients.
- An **RMI Activation Daemon**, which takes care of managing the states of services, for instance restarting crashed services, or deactivating and reactivating services based on whether they're being used or not (these responsibilities are described in more detail below).

- A **Lookup Service**, which allows clients to look up and find the Jini services that are currently available on the local network.
- A **Transaction Manager**, which is needed if your JavaSpaces applications make use of transactions.

Once these services are up and running, you can start:

- A **JavaSpaces Service**.

When building and testing JavaSpaces applications, you'll most likely run these services and your JavaSpaces client programs on a single desktop machine, but when deploying your final code, you'll most likely run them in a multimachine environment. So, let's step through the details of starting each of these services, paying attention along the way to avoiding pitfalls that could occur as you move your code from testing into deployment.

Starting an HTTP Server

Web servers are used for delivering class code to Jini clients. Let's take a brief look at how this works.

Whenever you run a JavaSpaces application that needs to serve classes to clients (for example, one that writes entries into a JavaSpace), you need to include "codebase" information on the command line (as you'll see in Running JavaSpaces Programs). The codebase specifies locations--usually URLs--from which a client can download the class files. Whenever your code writes entry objects into the space, the entries get annotated with that information. When a client of the space later removes or reads an entry object from the space, the entry's codebase annotation tells the client where to look for the class code, which is downloaded from an HTTP server (if the class doesn't already exist in the classpath). Typically you'll run a web server on each machine that needs to export code.

Any web server that's capable of serving documents can do the job, but for convenience, the Jini distribution provides a simple HTTP server in the tools.jar package. Here is the command to start the supplied HTTP server

(where optional parameters are shown in square brackets):

```
java -jar C:\jini1_0_1\lib\tools.jar
    [-port port-number]
    [-dir document-root-dir]
    [-verbose]
```

For example, on the Windows platform, you might issue this command:

```
java -jar C:\jini1_0_1\lib\tools.jar
    -port 8081
    -dir C:\jini1_0_1\lib
    -verbose
```

On Solaris, the equivalent command is:

```
java -jar /jini1_0_1/lib/tools.jar
    -port 8081
    -dir /jini1_0_1/lib
    -verbose
```

The JAR file containing the HTTP Server is C:\jini1_0_1\lib\tools.jar; this is an "executable JAR" file (introduced in the Java 2 platform), which means that it starts a default program (in this case, the HTTP server) when it's invoked as shown.

Here, you started the HTTP Server at port 8081 (if left unspecified, the port number would be 8080). The -dir option specifies which directory the server uses as its "document root." Here, you specified the Jini lib directory as the root, so this HTTP server is able to serve all the Jini JAR files in that directory (as well as other classes you might put there). The -verbose option causes information to be displayed about each request to the server, and is useful for debugging purposes.

Starting an RMI Activation Daemon

An RMI Activation Daemon is needed by several other services--the Transaction Manager, the persistent JavaSpaces Service, and the Jini Lookup Service--and an instance of the daemon needs to be run on each of the machines where you run these services. What does an activation daemon do? Suppose you have a service that you'd like to be automatically restarted if it crashes (for instance, if the machine it is running on crashes). Or, suppose you're interested in conserving computational resources, so you'd like a service to be brought down when it has no active clients and

reactivated later when a client tries to use the service. An activation daemon takes care of restarting, deactivating, and reactivating your services (such as the Jini services mentioned above).

Let's see how to start an RMI activation daemon. First, be sure your executable path points to the bin directory of JDK1.2.2; this is where rmid is located. Again, make sure that you haven't modified your CLASSPATH environment variable or placed JAR files in the Java extensions directory.

Then, to start rmid under Windows or Solaris, the command takes the form:

```
rmid [-port port-number] [-log dir]
```

For example, to start the RMI Activation Daemon on either Windows or Solaris, you could use the command:

```
rmid
```

to start the daemon on the default port of 1098. For further information on starting rmid, refer to the manual pages for Solaris

<<http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/rmid.html>>

or Windows

<<http://java.sun.com/products/jdk/1.2/docs/tooldocs/win32/rmid.html>>.

Starting a Lookup Service

A lookup service registers the Jini services available on the local network, and is used by applications to locate services such as a JavaSpace. You can currently choose to use either the RMI registry or the Jini lookup service to serve this purpose, but you should be aware that future versions of Jini services may not support the use of the RMI registry. Therefore, this document only covers using the Jini lookup service (if you really want to use the RMI registry, you can find the details in c:\jini1_0_1\doc\example\StartingService.html).

The command to start the Jini lookup service takes this form:

```
java -Djava.security.policy=security-policy-file
```

```
-jar C:\jini1_0_1\lib\reggie.jar  
lookup-codebase  
security-policy-file  
log-directory  
[lookup-group]
```

Let's examine each of the parameters.

The `-Djava.security.policy` option to `java` specifies a "security policy file" for the JVM to use when it runs the lookup service JAR file. The concept of a security policy file is new in the Java 2 platform; it is used to specify what resources a program can use when it runs--for example, whether it is permitted to access system properties, create and listen to socket connections, write to log files, and so on--and what level of privileges to grant to any downloaded code.

The JAR file containing Sun's implementation of the lookup service (called "reggie") is `C:\jini1_0_1\lib\reggie.jar`. Once again, this is an executable JAR file, which means it will start running (in a "setup JVM") when it's invoked as shown. First the remaining parameters are parsed, then the reggie lookup service is registered with the RMI Activation Daemon you started earlier on the same machine, and then the setup JVM exits (don't be alarmed that your command returns at this point, and that your DOS window, under Windows, says "Finished"). The lookup service is activatable: It will be restarted automatically by the activation daemon after crashes, and it will shift between "active" and "inactive" states as appropriate. In this case, the activation daemon spawns a new "server JVM" to run the lookup service; in fact, whenever a lookup service is activated, a new "server JVM" is spawned to run that instance of the service.

The final four parameters deserve a closer look. The `lookup-codebase` option specifies a URL that points to the `reggie-dl.jar` file, which contains the code that clients need to download to use the lookup service. For example, you might specify a URL to the HTTP server you started earlier, that is, `http://hostname:port/reggie-dl.jar`, where `hostname` is the name or IP address of the machine on which the HTTP server is running, and `port` is the port number at which it is running. You also need to ensure that the specified HTTP server can access `reggie-dl.jar` under its document root directory. To test to make sure the JAR file is accessible, you

can try accessing it from a browser, using the codebase URL. Since you started the web server in verbose mode, you should see a request logged to its screen, and you'll most likely see a "Save To" dialog box (unless the browser has been configured to handle .jar files).

The next option, *lookup-policy-file*, specifies another security policy file. The first security policy file you specified is used only by the setup JVM, while this second (potentially different) security policy file is used by the server JVM whenever the lookup service is reactivated.

The next parameter, *log-directory*, specifies an absolute path name to a directory (on the file system where the lookup service is running) where the lookup service writes its logs. It's important to point out that this directory should not already exist, or the setup JVM will exit without registering the lookup service.

The final, optional parameter, *lookup-group*, is used to specify a "group" of services for which reggie provides lookup service. Usually, you use the special group name *public*, which Jini services conventionally join; if you omit the parameter, the default is *public*.

Here is an example command for starting the reggie lookup service on Windows:

```
java -Djava.security.policy=C:\jini1_0_1\example\lookup\policy.all
-jar C:\jini1_0_1\lib\reggie.jar
http://hostname:8081/reggie-dl.jar
C:\jini1_0_1\example\lookup\policy.all
C:\tmp\reggie_log
public
```

where *hostname* is the name or IP address of the machine on which the HTTP server is running.

Here is what the example command looks like on the Solaris platform:

```
java -Djava.security.policy=/jini1_0_1/example/lookup/policy.all
-jar /jini1_0_1/lib/reggie.jar
http://hostname:8081/reggie-dl.jar
/jini1_0_1/example/lookup/policy.all
/tmp/reggie_log
public
```

You should note that using the *policy.all* file as shown is fine when you're experimenting with your code, but it's not recommended for production use, since it grants all permissions to all code (trusted or not).

When deploying your code in a production environment, make sure you've devised careful policies that restrict access appropriately (for instance, your policy might state that downloaded code cannot access certain parts of your file system). The topic of security in the Java 2 platform is beyond the scope of this article.

Starting a Transaction Manager

If your JavaSpaces applications make use of transactions, you'll need to start a transaction manager service next, with a command of the form:

```
java -Djava.security.policy=security-policy-file
-Dcom.sun.jini.mahalo.managerName=txnmanager-name
-jar C:\jini1_0_1\lib\mahalo.jar
txnmanager-codebase
security-policy-file
log-directory
[lookup-group]
```

(Note that, if you're using an RMI registry instead of the Jini Lookup, you'll need to modify the command slightly; refer to C:\jini1_0_1\doc\example\StartingService.html.)

The executable JAR file containing Sun's implementation of the Transaction Manager Service (called "mahalo") is C:\jini1_0_1\lib\mahalo.jar. Like the lookup service, the transaction manager service is activatable: When the command above is run, a "setup JVM" registers the transaction manager with the RMI activation daemon--which activates the manager in a newly spawned "server JVM"--and then exits. Once again, the command line returns once the registration is complete, and you should see a command prompt.

The *security-policy-file* options mean what you would expect, given our previous discussion about starting the Jini Lookup Service: The *-Djava.security.policy* option governs the permissions the setup JVM uses, while the second *security-policy-file* option specifies the permissions that the server JVM uses whenever the Transaction Manager Service is activated.

The option *-Dcom.sun.jini.mahalo.managerName* allows you to specify a name (such as "TransactionManager") for the transaction manager if you wish: this name becomes an attribute attached to the transaction manager in the

lookup service, and can be used to look up this particular transaction service later on.

Just as you did with the lookup service, you need to supply a codebase option to the service, specifying a URL to the mahalo-dl.jar file, which contains the code that clients need to download to use the transaction manager service. For example, you might supply a URL pointing to the hostname and port number of the HTTP server you started earlier. Be sure that the specified HTTP server can access mahalo-dl.jar under its root directory. Once again, you may want to try accessing the JAR file using a web browser and your codebase URL, to make sure the file is accessible.

The next parameter, *log-directory*, should specify an absolute path name to a directory (on the file system where the Transaction Manager service is running) where the service writes its logs. You should make sure that the log file doesn't already exist.

For the last (optional) parameter, you can specify the name of a Jini group for the Transaction Manager to join; for example, *public* means the manager should become part of the public group.

Here's an example of starting a Transaction Manager, using the Jini Lookup service, on Windows:

```
java -Djava.security.policy=C:\jini1_0_1\example\txn\policy.all
-jar C:\jini1_0_1\lib\mahalo.jar
http://hostname:8081/mahalo-dl.jar
C:\jini1_0_1\example\txn\policy.all
C:\tmp\txn_log
public
```

where again, *hostname* is the name or IP address of the machine on which the HTTP server is running. On Solaris, the command looks like this:

```
java -Djava.security.policy=/jini1_0_1/example/txn/policy.all
-jar /jini1_0_1/lib/mahalo.jar
http://hostname:8081/mahalo-dl.jar
/jini1_0_1/example/txn/policy.all
/tmp/txn_log
public
```

Starting a JavaSpaces Service

Now that you've started all the supporting services (HTTP Server, RMI Activation Daemon, Jini Lookup

Service, and Transaction Manager Service), you can finally start a JavaSpaces service (Sun's implementation is known as "outrigger"). You can choose to start either a JavaSpaces service called TransientSpace (one that doesn't maintain data across crashes and restarts of the space) or a persistent JavaSpaces service called FrontEndSpace (one that does maintain data across crashes and restarts).

Starting a Transient JavaSpace

To start a TransientSpace JavaSpaces service, given that you're running a Jini Lookup Service, the command takes the following form:

```
java -Djava.security.policy=security-policy-file
      -Djava.rmi.server.codebase=javaspace-codebase
      -Dcom.sun.jini.outrigger.spaceName=space-name
      -jar C:\jini1_0_1\lib\transient-outrigger.jar
      [lookup-group]
```

(Refer to C:\jini1_0_1\doc\example\StartingService.html for the details of formatting the command if you're using an RMI registry instead.)

Let's examine the parameters. The *security-policy-file* option governs the permissions the JVM uses when running the JavaSpaces service. The *javaspace-codebase* option specifies a URL that points to the *outrigger-dl.jar* code that clients must download in order to use the JavaSpaces service. The *-Dcom.sun.jini.outrigger.spaceName* option is used to specify a name for the JavaSpaces service (for example, "JavaSpaces"). And as you saw with the transaction manager service, you can optionally specify the name of a Jini community for this service to join.

It's worth noting here that a transient JavaSpace service is not an activatable service, and the command above is handled by just a single JVM. This is why you don't need to pass additional arguments (*codebase*, *security-policy-file*, *log-directory*) at the end of the command as you did for the lookup service and transaction manager service. The JVM that executes the command will continue to run until the JavaSpaces service exits or gets killed, so unlike the activatable services, you will not be returned to the command prompt while the space is running.

Here's an example of starting a transient JavaSpace, using the Jini Lookup service, on Windows:

```
java -Djava.security.policy=C:\jini1_0_1\example\books\policy.all
-Djava.rmi.server.codebase=http://hostname:8081/outrigger-dl.jar
-Dcom.sun.jini.outrigger.spaceName=JavaSpaces
-jar C:\jini1_0_1\lib\transient-outrigger.jar
public
```

On the Solaris platform, the equivalent command looks like this:

```
java -Djava.security.policy=/jini1_0_1/example/books/policy.all
-Djava.rmi.server.codebase=http://hostname:8081/outrigger-dl.jar
-Dcom.sun.jini.outrigger.spaceName=JavaSpaces
-jar /jini1_0_1/lib/transient-outrigger.jar
public
```

Again, you should note that the policy.all file is fine for your code experimentation, but you should revisit the security policy issue when you deploy your code in a production environment.

Starting a Persistent Javaspaces

In a deployment environment, you're likely going to want to run a persistent FrontEndSpace JavaSpaces service rather than a transient JavaSpaces service. A FrontEndSpace is able to maintain its data, even through crashes and restarts of the space.

To start a FrontEndSpace JavaSpaces service, given that you're running a Jini Lookup Service, the command takes the following form:

```
java -Djava.security.policy=security-policy-file
-Dcom.sun.jini.outrigger.spaceName=space-name
-jar C:\jini1_0_1\lib\outrigger.jar
javaspace-codebase
security-policy-file
log-directory
[lookup-group]
```

Since FrontEndSpace is an activatable JavaSpaces service, you'll notice some differences from starting a transient space. First, the codebase is specified differently. Second, you must supply a second security policy file (as you did when starting the lookup service and the transaction manager, to be used when the service is reactivated by the RMI Activation Daemon). Last, you must supply an absolute path name to a directory where the JavaSpaces service writes log files.

Note that outrigger.jar contains all the classes needed for the JavaSpaces service to run a FrontEndSpace. Here's an example of starting a FrontEndSpace, using a Jini Lookup service, on the Windows platform:

```
java -Djava.security.policy=C:\jini1_0_1\example\books\policy.all
-jar C:\jini1_0_1\lib\outrigger.jar
http://hostname:8081/outrigger-dl.jar
-Dcom.sun.jini.outrigger.spaceName=JavaSpaces
C:\jini1_0_1\example\books\policy.all
C:\tmp\js_log
public
```

Here's an example of starting a FrontEndSpace on Solaris:

```
java -Djava.security.policy=/jini1_0_1/example/books/policy.all
-jar /jini1_0_1/lib/outrigger.jar
http://hostname:8081/outrigger-dl.jar
-Dcom.sun.jini.outrigger.spaceName=JavaSpaces
/jini1_0_1/example/books/policy.all
/tmp/js_log
public
```

As you saw with other activatable services, issuing these commands returns a command prompt.

Compiling JavaSpaces Programs

By now, you have all the required Jini services up and running. Assuming you've already written a JavaSpaces client program (one that makes use of a JavaSpaces service), all that's left is to compile your code and run it.

If you wish, you can experiment with a very simple "Hello World" program from *JavaSpaces Principles, Patterns and Practice*. You need to edit the compile.bat file and replace localhost:8081 with the IP address and port number of your web server, and then you can simply run compile.bat and runit.bat to get the example up and running under Windows.

When compiling JavaSpaces programs, the javac compiler (and later the java bytecode interpreter) may need access to several different jar files:

- jini-core.jar, which contains the core Jini interfaces and classes and is needed whenever your program makes use of leases or events, for example.
- jini-ext.jar, which contains additional interfaces that are useful in building Jini applications, including the JavaSpaces interface.

- sun-util.jar, which contains a set of useful helper utilities. These classes may change, however, since they are regarded as non-standard.
- space-examples.jar, which contains various JavaSpaces technology examples and includes the classes com.sun.jini.mahout Locator, com.sun.jini.outtrigger.LookupFinder, and com.sun.jini.outtrigger.Finder, which are needed when compiling the SpaceAccessor.java code found in Chapter 1 of the *JavaSpaces™ Principles, Patterns and Practice* <<http://www.amazon.com/exec/obidos/ASIN/0201309556/>>.

When compiling your JavaSpaces programs, supply a -classpath option to the javac compiler. For example, to compile ExampleProgram.java on Windows, the command might look like this:

```
javac -d . -classpath
C:\jini1_0_1\lib\jini-core.jar;
C:\jini1_0_1\lib\jini-ext.jar;
C:\jini1_0_1\lib\sun-util.jar;
C:\jini1_0_1\lib\space-examples.jar;.;
ExampleProgram.java
```

Note that these JAR files might not all be necessary for every compilation; it depends on the specifics of the program you're compiling. Also note the use of the -d option; here it is used to specify that class files should be created in the current directory. The .; appearing at the end of the classpath option indicates that the current directory is searched for classes, along with the JAR files listed. You should tailor these options according to where you want your class files stored.

On Solaris, the equivalent command looks like this:

```
javac -d . -classpath
/jini1_0_1/lib/jini-core.jar:
/jini1_0_1/lib/jini-ext.jar:
/jini1_0_1/lib/sun-util.jar:
/jini1_0_1/lib/space-examples.jar::
ExampleProgram.java
```

Alternately, you could specify the required JAR files in the CLASSPATH environment variable or place JAR files in the Java extensions folder. However, during the code development phase--when you're likely running a JavaSpaces service and client programs on the same machine--avoiding these modifications to the classpath environment is a good policy: It helps to avoid unintended sharing of class files, and to simulate a deployment environment--when Jini services, such as a JavaSpace, and client programs that make use of them

are likely to run on different machines. Avoiding use of the CLASSPATH and Java extensions can help to reduce the problems you'll encounter when you move from development to deployment.

Running JavaSpaces Programs

Now that you've compiled your JavaSpaces programs, you're ready to run them. Assuming you're running the Jini Lookup service on the Windows platform, the command takes this form:

```
java -Djava.security.policy=C:\jini1_0_1\example\books\policy.all
-Doutrigger.spacename=JavaSpaces
-Dcom.sun.jini.lookup.groups=public
-cp C:\jini1_0_1\lib\space-examples.jar;
    C:\jini1_0_1\lib\jini-core.jar;
    C:\jini1_0_1\lib\jini-ext.jar;
-Djava.rmi.server.codebase=http://hostname:8081/space-examples-dl.jar
ExampleProgram
```

On Solaris, the command takes this form:

```
java -Djava.security.policy=/jini1_0_1/example/books/policy.all
-Doutrigger.spacename=JavaSpaces
-Dcom.sun.jini.lookup.groups=public
-cp /jini1_0_1/lib/space-examples.jar:
    /jini1_0_1/lib/jini-core.jar:
    /jini1_0_1/lib/jini-ext.jar:
-Djava.rmi.server.codebase=http://hostname:8081/space-examples-dl.jar
ExampleProgram
```

Let's take a look at each of the parameters. The `-Djava.security.policy` option specifies the path to the security policy file you'd like your program to use when it runs downloaded code (for example, if your program retrieves an entry from a JavaSpace and invokes one of its methods). If you find none of the policy files provided in the Jini distribution to be suitable, you'll have to modify one to meet your needs.

With the `-Doutrigger.spacename` option, you can specify the name of the specific JavaSpaces service you'd like your program to use (with the command above, you're specifying that your program should interact with a space called "JavaSpaces", if one can be found, not one called "Old JavaSpace" or anything else).

The `-Dcom.sun.jini.lookup.groups` option specifies which Jini community this program runs under (here, the public group).

The Java bytecode interpreter (java) needs access to several JAR files; the exact ones depend on what your program does. Refer to the list of JAR files in Compiling JavaSpaces Programs.

The codebase property needs to be set if your program is exporting downloadable code, for instance if your program is writing entries into a JavaSpace. In this case, when other programs read or remove those entries, they need to know where they can download the classes for those entries. This information is provided by your program, which annotates the entries it writes into the space with information from the codebase property. The codebase property also needs to be set if your program is requesting notification services from any of the Jini services; in this case the "listener" code needs to be exported.

LIST OF REFERENCES

- [BER99] Naval Postgraduate School Report NPSCS-00-001, *Interoperability Technology Assessment for Joint C4ISR Systems*, by Valdis Berzins, Luigi, Bruce Shultes, Jiang Guo, Jim Allen, Ngom Cheng, Karen Gee, Tom Nguyen, and Eric Stierna, September 1999.
- [CAR89] Nicholas Carriero, and David Gelernter, "How to Write Parallel Programs: A Guide to the Perplexed" ACM Computing Surveys, September 1989, pp.102-122.
- [GEL85] David Gelernter, "Generative Communication in Linda", ACM Transaction on Programming Languages and Systems, Vol. 7, No. 1, January 1985, pp. 80-112.
- [JOY99] Bill Joy, *The Jini Specification*, Addison Wesley, Inc., 1999.
- [KEI99] Edward Keith, *Core Jini*, Prentice Hall, PTR, 1999.
- [KIM95] Eun-Gyung Kim, "A Study on Developing a Distributed Problem Solving System" IEEE Software, January 1995, pp. 122-127.
- [KUHN99] Fred Kuhns, Carlos O'Ryan, Douglas Schmidt, Ossama Othman, and Jeff Parsons, "The Design and Performance of a Pluggable Protocols Framework for Object Request Broker Middleware", IFIP 6th International Workshop on Protocols For High-Speed Networks (PfHSN' 99), August 25-27, 1999.
- [LEV00] David Levein, Sergio Flores-Gaitan, and Douglas Schmidt, "An Empirical Evaluation of OD Endsysten Support for Real-time CORBA Object Request Brokers", Multimedia Computing and Network 2000, January 2000.

- [LUQ88] Luqi, and Valdis Berzins, "Rapidly Prototyping Real-Time Systems", IEEE Software, September 1988, pp. 25-35.
- [LUQ89] Naval Postgraduate School, *A Proposed Design for a Rapid Prototyping Language*, by Luqi, Valdis Berzins, Bernd Kraemer, and Laura White, March 1989
- [LUQ92] Luqi, Mantak Shing, "CAPS - A Tool for Real-Time System Development and Acquisition", Naval Research Review, Vol 1, 1992, pp.12-16.
- [LUQ98] Luqi, Valdis Berzins, and Raymond Yeh, "A Prototyping Language for Real-Time Software", IEEE Software, October 98, pp.1409-1423.
- [SUL99] Kevin Sullivan, Mark Marchukov, and John Socha, "Analysis of a Conflict Between Aggregation and Interface Negotiation in Microsoft's Component Object Model" IEEE Transactions on Software Engineering, Vol. 25, No. 4, July/August 1999, pp. 584-599.
- [WOL89] Antoni Wolski, "LINDA: A System for Loosely Integrated DataBases", IEEE Software, January 1989, pp. 66-73.
- [XUL89] Andrew Xu, and Barbara Liskov, "A Design of a Fault-Tolerant, Distributed Implementation of Linda", IEEE Software January 1989, pp. 199-206.
- [YAN99] Jingshuang Yang, and Gail Kaiser, "JPernLite: Extensible Transaction Services for the WWW", IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 4, July/August 1999, pp. 639-657.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..... 2
8725 John J. Kingman Rd., STE 0944
Ft. Belvior, Virginia 22060-6218
2. Dudley Knox Library..... 2
Naval Postgraduate School
411 Dyer Road
Monterey, California 93943-5101
3. Chairman, Code CS..... 1
Naval Postgraduate School
Monterey, California 93943-5101
4. Dr. Valdis Berzins, Code CS/BE..... 2
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5101
5. Dr Luigi, Code CS/LQ..... 1
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5101
6. Dr. Swapan Bhattacharya..... 1
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5101
7. Ref Delgado..... 1
Defense Advanced Research Projects Agency
3701N, Fairfax Drive
Arlington, VA 22203-1714
8. Dr. David Hislop..... 1
US Army Research Office
Mathematical & Computer Science Divison
4300 Miami Blvd.
Research Trangle Park, NC 27709
9. Col John A. Hamilton..... 1
Office of Chief Engineer
Space & Naval Warfare Systems Command
4301 Pacific Highway
San Diego, CA 92110-3127

10. Mr. Cheng Heng Ngom 3
Blk 75, #08-132,
Bedok North Road
Singapore, Singapore 460075