Appendix C: Tools and Specifications Used in the Implementation

(i). Java

Java is a programming language developed by James Gosling at Sun Microsystems. It was first released in 1995 as a core component of Sun Microsystems' Java platform. The language is quite similar to C and C++. Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box projects. He aimed to implement a virtual machine and a language that had a familiar C/C++ style of notation.

Sun released the first public implementation as Java 1.0 in 1995. It promised "Write Once, Run Anywhere" (WORA), providing no-cost run-times on popular platforms. Major web browsers soon incorporated the ability to run Java applets within web pages, and Java quickly became popular. With the advent of Java 2 (released initially as J2SE 1.2 in December 1998), new versions had multiple configurations built for different types of platforms. For example, J2EE targeted enterprise applications and the greatly stripped-down version J2ME for mobile applications. J2SE designated the Standard Edition. In 2006, Sun renamed new J2 versions as Java EE, Java ME, and Java SE.

Java is one of the most popular languages because it is simple and supports object oriented programming paradigm. Java code is also considered to be safe as it provides security on several different levels. First, the language was designed to make it extremely difficult to execute damaging code. The elimination of pointers is a big step in this regard. Another level of security is the bytecode verifier. Java programs are compiled into a set of instructions called bytecodes. Before a Java program is run, a verifier checks each bytecode to make sure that nothing suspicious is going on. Java is also known for its platform independent code.
Java supports numerous Application Programming Interface (API). The API is a set of classes used to develop Java programs. These classes are organized into groups called packages. There are packages for the following tasks:

- Numeric variable and string manipulation
- Image creation and manipulation
- File input and output
- Networking
- Windowing and Graphical User Interface (GUI) design
- Applet programming
- Error handling
- Security
- Database access
- Distributed application communication
- JavaBeans components

During the implementation of MASICC, API that supports GUI design has been used. Swing and Abstract Window Toolkit (AWT) packages are used to create user interfaces for agents. Swing provides a more sophisticated set of GUI components than the AWT. Swing provides a native look and feel that emulates the look and feel of several platforms. The Swing architecture is characterized by the following features:

- Platform independence
- Component-oriented
- Customizable
- Lightweight UI
- Extensibility
- Look and feel
- Configurable
(ii). JADE

JADE (Java Agent DEvelopment Framework) is a software framework to develop agent-based applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. JADE can be considered an agent middleware that implements an Agent Platform and a development framework. It deals with all those aspects that are independent of the applications, such as message transport, encoding and parsing, or agent life-cycle. It is a product of R&D branch of Telecom Italia Lab and Motorola.

JADE provides the programmers the following ready-to-use and easy-to-customize functionalities.

- A fully distributed system inhabited by agents, each running as a separate thread, potentially on different remote machines, and capable of transparently communicating with one another.
- Full compliance with the FIPA specifications.
- Efficient transport of asynchronous messages via a location-transparent API. The platform selects the best available means of communication and, when possible, avoids marshalling/unmarshalling java objects.
- Implementations of both white pages and yellow pages.
- A simple, yet effective, agent life-cycle management. When agents are created they are automatically assigned a globally unique identifier and a transport address which are used to register with their platform's white page service. Simple APIs and graphical tools are also provided to both locally and remotely manage agent life cycles, i.e. create, suspend, resume, freeze, thaw, migrate, clone and kill.
- Support for agent mobility.
- A set of graphical tools to support programmers when debugging and monitoring.
- Support for ontologies and content languages.
Appendix C

Figure C1 shows the main architectural elements of a JADE platform. A JADE platform is composed of agent containers that can be distributed over the network. Agents live in containers which are the Java process that provides the JADE run-time and all the services needed for hosting and executing agents. There is a special container, called the main container, which represents the bootstrap point of a platform: it is the first container to be launched and all other containers must join to a main container by registering with it.

![Diagram of JADE Platform](image)

**Figure C1**: Jade Container and Platforms.

**Compiling the Software and running the platform**

All the software relating to JADE can be downloaded from the JADE website at http://jade.tilab.com. The main distribution is composed of five primary archive files with the following content:

- *jadeBin.zip* contains only the pre-compiled JADE Java archive (jar) files in a ready to use state.
Appendix C

- **jadeDoc.zip** contains the documentation, including the Administrator and Programmer guides. This documentation is also available online from the website.
- **jadeExamples.zip** contains the source code of various examples.
- **jadeSrc.zip** contains all the sources of JADE.
- **jadeAll.zip** contains all of the four files listed above.

The directory structure of these zipped files is shown in Figure C2.

![Figure C2: Directory structure of JADE](image)

Now, assume that JADE has been downloaded into C:\jade on a Windows platform, the CLASSPATH can typically be set using the following command-line sequence:

```
prompt> set JADE_HOME=c:\jade
prompt> set CLASSPATH=%JADE_HOME%/lib\jade.jar; %JADE_HOME%/lib\jadeTools.jar; %JADE_HOME%/lib\http.jar; %JADE_HOME%/lib\iiop.jar; %JADE_HOME%/lib\commons-codec\commons-codec-1.3.jar;%JADE_HOME%/classes
```

The main-container can now be launched with the JADE GUI using the command:

```
prompt> java jade.Boot -gui
```

The discussion about the JADE in this appendix is designed to help others get acquainted with this technology.
(iii). GeNLe and SMILE:

The rapid development of Bayesian network research over the past 15 years has been accompanied by a proliferation of BN software tools. These tools have been built to support both these research efforts and the applications of BNs to an ever-widening range of domains. GeNLe is a product of Decision Systems Laboratory, University of Pittsburgh. It is developed by Druzdzel’s decision systems group. GeNLe 1.0 was released in 1998, and GeNLe 2.0 is released in 2003. GeNLe (Graphical Network Interface) is a development environment for building decision networks, running under Windows. A snapshot is shown in Figure C3.

![GUI of GeNLe for constructing BN](image)

**Figure C3:** GUI of GeNLe for constructing BN.

This figure shows the GUI of GeNLe. It can be used to create and test the BN, graphically by setting the evidence(s) of node(s). The overall effect of evidence can then be observed by running the network.
SMILE (Structural Modeling, Reasoning and Learning Engine) is GeNIe’s portable inference engine, consisting of a library of C++ classes currently compiled for Windows, Solaris and Linux. GeNIe is an outer shell to SMILE. It supports conditional probability table of chance nodes. The graphical representation of probability distribution is of immense use.

The default BN inference algorithm of GeNIe is the junction tree clustering algorithm; however a polytree algorithm is also available, plus several approximate algorithms that can be used if the networks get too large for clustering. GeNIe 2.0 provides more recent state-of-the-art sampling algorithms. It is also capable of handling specific evidence. The main decision-theoretic modeling features supported by GeNIe are as follows:

- **Decision analysis**: It is based on the assumption that humans are reasonably capable of framing a decision problem, listing possible decision options, determining relevant factors, and quantifying uncertainty and preferences, but are rather weak in combining this information into a rational decision.

- **Utility**: An integral element of all decision problems is the notion of preference. Very often, preference can be based on an objective quantity, such as material usage, factory output, or financial gain. However, decision problems involve quantities that have no obvious numerical measure, such as state of health, customer satisfaction, or pain.

- **Bayesian Network and Bayesian Updating**: Bayesian networks are acyclic directed graphs in which nodes represent random variables and arcs represent direct probabilistic dependences among them. More details can be found in section 3.4. Bayesian updating deals with the computing the impact of evidence(s) of a subset of the model variables on the probability distribution over the remaining variables.

In MASICC, we used GeNIe to imitate the behavior of a pediatrician, using Bayesian Network, during the selection of a medical specialist agent. One can refer to section 5.3 for more details about the designing issues of probabilistic network of the MASICC.
(iv). MATLAB

MATLAB is a technical computing environment developed by The Math-Works, Inc. for computation and data visualization. It is both an interactive system and a programming language, whose basic data element is an array: scalar, vector, matrix or multi-dimensional array. Besides basic array operations, it offers programming features similar to those of other computing languages (e.g., functions, control flow, etc.). MATLAB executes on Windows, UNIX, and Linux systems.

The main features of MATLAB are discussed below:

- It is an interactive system for doing numerical computations.
- It relieves you of a lot of the routine tasks associated with solving numerical problems.
- It allows you to spend more time thinking, and encourages you to experiment while writing few line of code.
- It makes use of highly respected algorithms (already built) and hence one can be confident about ones results.
- Powerful operations can be performed using just few commands.
- One can build up ones own set of functions for a particular application.
- Excellent graphics facilities are available, and the pictures can be inserted into Word documents.

One of the strengths of Matlab is that its commands match very closely to the steps that are used to solve real life problems; thus the process of determining the steps to solve the problem also determines the Matlab commands. Furthermore, Matlab includes extensive toolboxes, such as Fuzzy Toolbox, Neural Network Toolbox, Financial Toolbox, Image Acquisition Toolbox, Image Processing Toolbox, etc. This greatly reduces the programming effort.

The NN toolbox can be used for designing, implementing, visualizing, and simulating neural networks. Neural networks are invaluable for applications where
formal analysis would be difficult or impossible, such as pattern recognition and nonlinear system identification and control. Neural Network Toolbox software provides comprehensive support for many proven network paradigms, as well as graphical user interfaces (GUIs) that enable the user to design and manage networks. This is shown in Figure C4. It is the output of running the ‘nntool’ command on the command prompt. Some of the key features of this toolbox are:

- GUI for creating, training, and simulating neural networks
- Quick start wizards for fitting, pattern recognition, and clustering
- Support for the most commonly used supervised and unsupervised network architectures
- Comprehensive set of training and learning functions
- Simulink blocks for building neural networks and advanced blocks for control systems applications
- Support for automatically generating Simulink blocks from neural network objects
- Visualization functions and GUI for viewing network performance and monitoring the training process

![Neural Network Manager](image)

**Figure C4:** Neural Network Manager.
Appendix C

In MASICC, Neural Network toolbox is used to create BPNN and PNN. We also utilized the gensim function to generate the simulation of neural networks. Figures 7.10 and 7.11 are the outputs of 'gensim' function.

(v). FIPA

The Foundation for Intelligent Physical Agents (FIPA) is a body for developing and setting computer software standards for heterogeneous and interacting agents and agent-based systems. FIPA was founded as a Swiss not-for-profit organization in 1996 with the ambitious goal of defining a full set of standards for both implementing systems within which agents could execute (agent platforms) and specifying how agents themselves should communicate and interact.

It has large number of members which include several academic institutions and a large number of companies including Hewlett Packard, IBM, BT (formerly British Telecom), Sun Microsystems, Fujitsu and many more. A number of standards were proposed for agent communication. The most widely adopted of the FIPA standards are the Agent Management and Agent Communication Language (FIPA-ACL) specifications. Now days, FIPA comes under the ambient of IEEE Computer Society.

FIPA has played a crucial role in the development of agents standards and has promoted a number of initiatives and events that contributed to the development and uptake of agent technology. The key achievements of FIPA are as follows:

- A set of standard specifications supporting inter-agent communication and key middleware services.
- An abstract architecture providing an encompassing view across the entire FIPA2000 standards. This architecture is known as the Java Agent Services (JAS).
- A well-specified and much-used agent communication language (FIPA-ACL), accompanied by a selection of content languages (e.g. FIPA-SL) and a set of key
interaction protocols ranging from single message exchange to complex transactions.

- Several open source and commercial agent tool-kits. JADE is generally considered as the leading FIPA-compliant open source technology available today.
- Several projects outside FIPA such as the completed Agentcities project that created a global network of FIPA-compliant platforms and agent application services.
- An agent-specific extension of UML, known as AUML or Agent

The main task of FIPA is to standardize agent communication, sub-layering of agent communication and agent management. These are discussed below:

1.1 Agent Communication

The FIPA-ACL is based on ‘speech act theory’ which states that messages represent actions, or communicative acts – also known as speech acts or performatives. A simple example would be ‘My name is Amit’ which, when issued, informs the recipient with an item of information. The FIPA-ACL set of 22 communicative acts. Some of the most commonly used acts are inform, request, agree, not understood, and refuse. These capture the essence of most forms of basic communication. For an agent to be FIPA compliant, the agent must be able to receive any FIPA-ACL communicative act message and at the very least respond with a not-understood message if the received message cannot be processed. Based on these communicative acts, FIPA has defined a set of interaction protocols, each consisting of a sequence of communicative acts to coordinate multi-message actions, such as the contract net for establishing agreements and several types of auctions.

1.2 FIPA Sub-Layers

The FIPA communication mechanism can be divided into several sub-layers within the application layer of the classical OSI or TCP/IP stack. These are detailed as follows:
• **Sub-layer 1 (Transport):** In the FIPA-ACL layered protocol model, the lowest application sublayer protocol is the transport protocol. FIPA has defined message transport protocols for wireless application protocol and HTTP.

• **Sub-layer 2 (Encoding):** Rather than send simple byte-encoded messages, FIPA defines several message representations for using higher-level data structures including XML, String and Bit- Efficient.

• **Sub-layer 3 (Messaging):** The important aspect at this level is that key parameters that are necessary for content to be exchanged, e.g. the sender and receiver, the message type (communicative act), time-outs for replies, etc.

• **Sub-layer 4 (Ontology):** The individual terms contained in the content of a FIPA message can be explicitly referenced to an application-specific conceptual model or ontology.

• **Sub-layer 5 (Content expression):** The actual content of FIPA messages can be of any form, but FIPA has defined guidelines for the use of general logical formulas and predicates, and algebraic operations for combining and selecting concepts. The language most often used for expressing content is FIPA-SL.

• **Sub-layer 6 (Communicative act):** The simple classification of a message in terms of the action, or performative, that it implies.

• **Sub-layer 7 (Interaction protocol or IP):** Typically messages are rarely exchanged in isolation but rather form part of some interaction sequence. FIPA defines several interaction protocols specifying typical message exchange sequences such as request which describes one party making a request of another which in turn should agree or refuse to comply.

### 1.3 Agent Management

In addition to communication, the other fundamental aspect of agent systems addressed by the FIPA specifications is agent management: a framework within which FIPA compliant agents can exist, operate and be managed. It aims to manage: creation, registration, location, communication, migration and operation of agents. Agent Platform,
Agent, Directory Facilitator, Agent management system and Message transport service are the components of agent management. Brief descriptions about these are as follows:

**Agent Platform (AP):** This provides the physical infrastructure in which agents are deployed. The AP consists of the machines, operating systems, FIPA agent management components, the agents themselves and any additional support software.

**Agent:** An agent is a computational process that inhabits an AP and typically offers one or more computational services that can be published as a service description. An agent must have at least one owner and must support at least one notion of identity which can be described using the FIPA Agent Identifier (AID) that labels an agent so that it may be distinguished unambiguously.

**Directory Facilitator (DF):** The DF is an optional component of an AP providing yellow pages services to other agents. It maintains an accurate, complete and timely list of agents and must provide the most current information about agents in its directory. An AP may support any number of DFs which may register with one another to form federations.

**Agent Management System (AMS):** The AMS is a mandatory component of an AP and is responsible for managing the operation of an AP, such as the creation and deletion of agents, and overseeing the migration of agents to and from the AP. Each agent must register with an AMS in order to obtain an AID which is then retained by the AMS as a directory of all agents present within the AP and their current state.

**Message Transport Service (MTS):** The MTS is a service provided by an AP to transport FIPA-ACL messages between agents on any given AP and between agents on different APs.
Appendix C: Data used in Neural Networks

<table>
<thead>
<tr>
<th>Cardiologist</th>
<th>Pulmonologist</th>
<th>Surgeon</th>
<th>Gastroenterologist</th>
<th>Endocrinologist</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

The table contains data in neural network format.