USING INTELLIGENT AGENTS FOR
COMPLEX SOFTWARE SYSTEMS MAINTENANCE

A Thesis Presented to
The Faculty of the
Fritz J. and Dolores H. Russ
College of Engineering and Technology
Ohio University
In Partial Fulfillment
Of the Requirement for the Degree
Master of Science
by
Mikhail P Dobrynin

June, 2002

OHIO UNIVERSITY
LIBRARY
# TABLE OF CONTENTS

List of Tables ...................................................................................................................... v

List of Figures .................................................................................................................... vi

1. Introduction .................................................................................................................... 1
   1.1 Object and Scope of the Thesis ................................................................................ 3
   1.2 Related Work .......................................................................................................... 4
   1.3 Organization of the Thesis ..................................................................................... 5

2. Complex software systems maintenance ........................................................................ 6
   2.1 Upgrading versus Patching ..................................................................................... 8
   2.2 Current trends in Software Upgrading ..................................................................... 9
   2.3 Problems with Complex Software Systems Patching .............................................. 12
   2.4 How to cut the cost of complex software systems maintenance ............................. 14

3. Intelligent Agents and their Applications ...................................................................... 17
   3.1 Definition ............................................................................................................... 17
   3.2 Classification .......................................................................................................... 19
   3.3 How information agents work ............................................................................... 20
   3.4 Applications .......................................................................................................... 21
   3.5 Tools ....................................................................................................................... 22

4. PaTra (Patch Tracker) ................................................................................................... 24
   4.1 Delegating patch tracking to an agent ................................................................... 24
   4.2 Design of PaTra ...................................................................................................... 33
      4.2.1 Three-tiered architecture ............................................................................... 33
      4.2.2 The Knowbot Tier ......................................................................................... 34
      4.2.3 The Database Tier ........................................................................................... 35
4.2.4 The Interface Tier ........................................................................................................... 35
4.3 Implementation of PaTra ..................................................................................................... 36
  4.3.1 WebL as a tool to build the knowbot module ............................................................ 37
  4.3.2 Java Servlet for the User Interface .............................................................................. 39
4.4 Result Validation ................................................................................................................. 39
4.5 Security Implications of using PaTra ............................................................................... 41

5. Conclusions and Recommendations .................................................................................. 43
  5.1 Conclusions ....................................................................................................................... 43
  5.2 Future Work ....................................................................................................................... 44

References ................................................................................................................................. 46

Appendix A: Agent Construction Tools: Commercial Products ........................................... 49
Appendix B: Agent Construction Tools: Research Projects .................................................. 51
Appendix C: Patra.Webl – Implementation of a knowbot ....................................................... 55
Appendix D: Patra.pkb – Implementation of AnalyzePatches PL/SQL function ...................... 58
Appendix E: patra.java – Implementation of the User Interface ............................................ 60
Appendix F: Demonstrating the operation of PaTra ............................................................... 68
Appendix G: Software Validation Questionnaire for PaTra .................................................. 78
LIST OF TABLES

Table 3.1: Attributes of an Intelligent Agent ................................................................. 18
Table 3.2: Types of agents .............................................................................................. 20
Table 3.3: Overview of agent languages ......................................................................... 23
LIST OF FIGURES

Figure 2.1 – Software maintenance process ................................................................. 7
Figure 2.2 – Windows Update on-line Service ........................................................... 10
Figure 2.3 – Symantec's LiveUpdate ............................................................................ 11
Figure 2.4 – CatchUp on-line service .......................................................................... 11
Figure 2.5 – An example of a patch description. .......................................................... 13
Figure 2.6 – Using support web site for software systems maintenance .................... 15
Figure 3.2 – Categories of intelligent agents .............................................................. 19
Figure 3.3 – A part view of an agent topology ............................................................ 19
Figure 3.4 – A view of how Information Agent works .............................................. 21
Figure 4.1 – An example of Oracle ERP system configuration ................................. 25
Figure 4.2 – Patch search form on Metalink ............................................................... 27
Figure 4.3 – A list of patches on Metalink ................................................................. 28
Figure 4.4 – Configuration hierarchy of PaTra ............................................................ 29
Figure 4.5 – The execution cycle of PaTra ................................................................. 30
Figure 4.6 – PaTra’s Web interface ............................................................................ 32
Figure 4.7 – Three-Tiered architecture of PaTra ....................................................... 34
Figure 4.8 – Using WebL for web page parsing ......................................................... 38
Figure F.1 – A sample run of PaTra’s knowbot module ........................................... 68
Figure F.2 – A notification e-mail sent by PaTra ...................................................... 69
Figure F.3 – Patches are grouped by instances in PaTra ........................................... 70
Figure F.4 – PaTra’s web interface................................................................. 71
Figure F.5 – Changing a patch status and deleting patches using PaTra’s web interface 72
Figure F.6 – PaTra’s confirmation of patch deletions and updates ....................... 73
Figure F.7 – PaTra presents only those patches whose status is checked............. 74
Figure F.8 – Number of patches per page can be adjusted .............................. 75
Figure F.9 – A Readme file is retrieved directly from Metalink ....................... 76
Figure F.10 – A patch is downloaded directly from Metalink .......................... 77
Chapter 1

INTRODUCTION

Approximately 70 percent of the total cost of owning software is related to maintenance, and the other 30 percent is development cost [1][2][3]. The price of software maintenance is high, getting even higher for complex software systems like ERP (Enterprise Resource Planning). ERP systems consist of many modules that have to be maintained. For example Oracle ERP system has more than 90 modules that manage important operations, including product planning, purchasing, inventory management, interacting with suppliers, order tracking, human resources, financial planning, and accounting [4]. All modules of ERP system have to be maintained.

Both software producers and software consumers deal with the maintenance of ERP systems. Software producers have to support the systems they sell by creating patches and upgrades to the modules of the systems. Software consumers are responsible for keeping the systems up and running.

Companies that have deployed ERP systems always have a team of IT personnel responsible for the maintenance of the system. The list of their activities usually include:

1. **Keeping track of new patches.** New patches usually come out on a regular basis and are accessible through a vendor’s web site. For example Oracle has
a tech-support web site called Metalink, where all the patches can be browsed by categories, have descriptions and prerequisites.

2. **Determining what patches have to be applied to the system.** Not all new patches should be applied to the system. Some patches have prerequisites, which are requirements that the system must meet in order to apply a particular patch. Some patches are cumulative, that is they contain several patches inside. Usually any complex software system internally keeps track of the patches that have been applied.

3. **Downloading appropriate patches.** A FTP session has to be established, with user authentication if required. Patches that have to be applied to the system are downloaded to a local host.

4. **Applying patches.** The application of patches is complicated and it is difficult to generalize procedure. There are software systems developed specifically for patches application, for example Kintana. Kintana keeps track of the patches that have been applied to the system. It defines a workflow for each patch application, because application process may consist of several steps that have to be approved by different offices.

One of the activities that the maintenance team does on a regular basis is keeping track of new patches that have to be applied to the system. This is a rather time consuming process especially when there are several instances of the same system and sometimes even different versions that have to be maintained. Software systems based on intelligent agent technology can effectively be used to automate some of the maintenance
procedures. This thesis proposes to delegate the task of keeping track of patches to an intelligent software agent. This can save a lot of time for a maintenance team and therefore cut the cost of the system maintenance in general.

1.1 Object and Scope of the Thesis

This thesis deals with the design and implementation of a software agent called PaTra (stands for PAtch TRAcker). PaTra is capable of keeping track of patches that have been applied (old) and have to be applied (new) to the system. It uses Oracle’s support web site called Metalink that can provide a list of patches for a specific product and a platform. The agent queries this web site on a regular basis, for example daily, to get a list of patches for platforms and products that are specified in the agent’s configuration file. If PaTra finds any new patches it stores their information into its database and sends an e-mail notification (that can be disabled) to a system administrator or any designated person. Upon receiving the notification, the system administrator can go to PaTra’s web interface to take a look at the new patches and make a final decision whether these new patches have to be applied to the system or not.

Possible application of PaTra is rather broad. Although in its current implementation it can only be used with Oracle ERP system, with slight modifications it can also be used in other complex software systems or ERP systems that have a similar scheme of interaction between a software producer and a software consumer by means of
a support web site. Nowadays majority of software companies (if not all) have support web sites where they publish developed patches, fixes, and upgrades.

1.2 Related Work

During an extensive search of literature and periodic publications only a few articles related to the subject of this thesis were found. Some discuss the use of intelligent agents for software maintenance just in theory, others also offer some practical solutions.

Sharma and Capretz in their “Application maintenance using software agents” [5] describe a system for helping in the maintenance process of a software application. They developed a group of intelligent agents that worked together to aid in software maintenance by automatically informing the appropriate individuals of any changes that were made to an open-source Internet software application. The system they developed helps to maintain an open-source application only.

Richard Scott in his PhD dissertation “Agent-based Software Configuration Deployment” [6] at Colorado University defines and prototypes a software deployment framework called Software Dock. He states that maintenance is a part of software deployment process and proposes a generic deployment process algorithm that uses agents and illustrates the relationships among software deployment processes in general. He also describes the Deployable Software Description (DSD) format used to specify the deployment requirements of software systems.
Software systems that have been proposed so far that use intelligent software agents for software systems maintenance do not seem to be very practical and are not applicable to modern ERP systems and other complex software systems. This thesis on the other hand proposes a practical agent-based solution aimed at ERP systems maintenance and is able to perform some of the maintenance activities for a system administrator.

1.3 Organization of the Thesis

The goal of this thesis is the design and development of the prototype. Chapter 1 gives you a brief introduction of a problem domain and a short description of developed software. The second and the third chapters are dedicated to the background of the thesis. Chapter 2 talks about complex software systems maintenance, current trends in software upgrading and problems with complex software systems patching. Chapter 3 introduces intelligent agents: definition, classification, architecture, applications and the tools for their development. It also discusses why a software agent technology was chosen. Chapter 4 gives a detailed design, the underlying structure, and the components of PaTra. It also talks about development tools used in the project. The fifth chapter states what has been achieved, presents the advantages and the limitations of PaTra, and also has recommendations for improvements and future work.
Chapter 2

COMPLEX SOFTWARE SYSTEMS MAINTENANCE

Software maintenance is the term that is used to describe the various software engineering activities that occur once a software system has been delivered and installed [1]. Software engineering theory gives us four types of these maintenance activities [2][3][7][8][9]:

- **corrective maintenance** fixes problems resulting from errors in the system,
- **adaptive maintenance** results in modification to the software to accommodate changes to its external environment,
- **perfective maintenance or enhancements** extends the software system beyond its original functional requirements,
- **preventive maintenance or reengineering** is the prevention of software problems in existing software before the problems occur.

All these software maintenance activities involve both software producers and software consumers. Software producers participate in the software maintenance process by getting a feedback from software consumers and developing upgrades and patches to the supported software systems. Software consumer’s role in the software maintenance process is: giving the feedback (comments, suggestions for improvements, complaints
about the software) to the software producers, and applying patches and upgrades to the software systems (Figure 2.1).

However, the level of involvement of a software consumer in the software maintenance heavily depends on the complexity of a maintained software system. No matter how complex the software system is, software consumers or users always generate some feedback for the software producers. Then the software producers, in their turn, develop patches and upgrades to address the users’ complaints and recommendations. In case of simple software products software developers just create new versions or upgrades and users just reinstall or upgrade their products. In case of complex software systems software developers usually produce patches to fix minor problems or make some modifications. Users of these complex software systems usually have to apply the patches on their own and because application of patches can be a cumbersome process and a number of patches can be significant the level of involvement of software consumers in this case is rather high. For example, Visa makes use of 21 mainframe
computers to run its 50 million line transaction processing system. This system is updated as many as 20000 times a year [10]. In fact, companies that deploy complex software systems, like ERP, have IT support staff responsible for the maintenance of their systems.

2.1 Upgrading versus Patching

It is important at this point to make a distinction between an upgrade and a patch. Although these terms are similar there are significant differences. An upgrade implies that a version of a software system changes. Upgrades replace an old software system with a newer one. A patch, on the other hand, is the immediate solution that is provided to users. It can sometimes be downloaded from the software maker's Web site. The patch is not necessarily the best solution for the problem and the product developers often find a better solution to provide when they package the product for its next release. Application of a patch only partially changes software system, whose version usually stays the same. Usually patches are used when the cost associated with the development and applying of a patch is significantly less than the cost associated with the development and applying of another upgrade. That is why simple software almost never has patches, but instead users just upgrade it. On the other hand, complex software systems usually get upgraded once every few years, while being constantly patched. Large numbers of patches in a software system may contribute to instability and security problems though, that is why software developers try not to issue too many patches between software upgrades.
2.2 Current trends in Software Upgrading

The growing connectivity between software producers and software consumers has given software producers a new tool of leverage in their attempts to offer their customers high-level deployment services [11]. From the perspective of the software consumers, they want to purchase, install, maintain, and support their software systems via the Internet. As a result it is becoming increasingly difficult for software producers to develop their software systems without taking these issues into consideration [11]. The Internet offers new interaction possibilities between software producers and consumers. Many software companies already have different on-line services to support their products. Let's look at some examples. Microsoft Corp's on-line Windows Update service allows a user to check a local installation of Windows for available updates, security packs, and new device drivers (Figure 2.2). Symantec's LiveUpdate makes it easy to keep Norton Utilities up-to-date with the latest program updates (Figure 2.3). Many software products nowadays have a built-in functionality to check for availability of a newer version using the Internet connection. One good example is WinAmp that can check for a new version every time it is run. Some sites even offer an independent, all-system check-up for upgrades and patches. One such service is CatchUp.com. The CatchUp software scan simultaneously searches PC for installed versions of software applications and hardware drivers. The scan results page shows the components that CatchUp located on the PC that are supported by the service. Then a user can select
software components to be updated and the CatchUp software downloads selected patches and updates via Internet and installs them on a PC (Figure 2.4).

Figure 2.2 – Windows Update on-line Service
Figure 2.3 – Symantec's LiveUpdate

Figure 2.4 – CatchUp on-line service
Thus, as we can see software development companies try to use the benefits that the Internet provides for high-level software maintenance to meet growing consumer expectations.

Ideal software is one that keeps itself updated. There are a number of interesting researches on dynamic software updating [10] and Configurable Distributed Systems [11] that run continuously or for very long periods of time. Perhaps in the near future we will see these theories implemented in marketed software systems.

2.3 Problems with Complex Software Systems Patching

Complex software systems, such as ERP systems, require more sophisticated maintenance procedures than simple software systems. There are several reasons for this:

- complex systems are constantly changing,
- a lot of patches to be tracked,
- there is no general procedure to apply patches.

The last statement represents probably the biggest problem in complex software systems patching. In software systems like Windows, for example, there is a general procedure to apply patches. A patch in such a system is basically a set of operations that may include:

- file deletion,
- file creation,
- file replacement (usually with a newer version),
- modifications in a Windows registry,
- restart of the operating system.

The existence of a general procedure to apply patches enabled Microsoft to automate this process (Figure 2.1), and not just automate, but do it very conveniently for users – through Microsoft’s Windows Update on-line service.

Unfortunately, many complex software systems, like ERP, do not have a general procedure for application of patches. Each patch in these systems is accompanied by a patch description or Readme file (see Figure 2.5), which contains information about the

```sh
# Copyright (C) 1993-2000 Oracle Corporation, California, USA
#
# This is a patch for Bug 1366454 to TG4DRDA 8.0.4.1.0 for AIX
#
# Pre-Requisites: none
# Supercedes: none
# Updates: none
#

The Patch is in compress TAR format. To install:

1) Set your ORACLE_HOME to point to the install gateway home
   export ORACLE_HOME=<gateway_install_home>

2) Change directory to the lib directory
   cd $ORACLE_HOME/tg4drda/lib

3) Uncompress and Untar the downloaded patch
   uncompress Bug1366454.tar.Z
   tar xvf Bug1366454.tar

   You should receive three files in the tar packet:
   README.1366454
   hgdrda.o_dbg
   hgdrda.o_prd

4) To apply the fix, simply run "sh README.1366454"

5) You will need to remake the g4drsrv executable.
   cd $ORACLE_HOME/tg4drda/lib
   make -f tg4drda.mk g4drsrv
```

Figure 2.5 – An example of a patch description.
These patch descriptions can be very technical. That is why only people with broad knowledge and understanding of the system can deal with complex software systems patching.

Thus, the complexity of a patch application process makes it necessary to have a team of professionals to maintain the system and therefore increases the cost associated with the software maintenance in general.

2.4 How to cut the cost of complex software systems maintenance

Both software producers and software consumers struggle to lower the cost of software maintenance. One of the steps on this path that has already been taken by many software companies is using the Internet as a tool to interact with customers by means of a support web site. Software companies publish patches to their products on their web sites. On these sites users can create requests for assistance (on Oracle’s support site
‘Metalink’, these requests are called TARs – Technical Assistance Requests), search for new patches, download them, and have access to extensive documentation libraries and knowledge bases (Figure 2.6). This increases the quality of customer support and lowers the cost of software maintenance.

There is more that can be done to lower this cost even further. A list of activities that software system maintenance personnel on a software consumer’s site usually do is:

- Keeping track of new patches,
- Determining what patches have to be applied to the system,
- Downloading appropriate patches,
- Applying patches.

By automating any of these activities the cost of software maintenance can be lowered. Keeping track of patches can effectively be automated by using intelligent agent technology discussed in the next chapter. PaTra, a software agent, automates a task of
keeping track of patches, thus allowing system administrators to concentrate on the other three tasks.

By adding a little intelligence to the agent and some collaboration on Oracle's part it is possible to automate the second activity - determining what patches have to be applied to the system. Currently Oracle's patch descriptions are very unstructured, and this makes it a very difficult task to make any automated decisions based on the descriptions. If Oracle used structured patch descriptions, preferably XML-based, it would be possible to automate the process of making a decision whether a particular patch has to be applied or not. If the problem of automating decision-making were solved, then it is pretty easy to automate the next activity – downloading patches that have been approved at the previous step. Automation of the last maintenance activity – applying patches – will pose a significant challenge until a general procedure for patch applications that was mentioned earlier in this chapter is established.

Thus, there are ways to further lower the cost of software maintenance and this thesis suggests one way to do that.
Chapter 3

INTELLIGENT AGENTS AND THEIR APPLICATIONS

3.1 Definition

Despite the fact that the term Agent is widely used there is no commonly accepted definition of what an agent is [11][12][13][14][15][16][17][18].

"Some have tried to offer the general definition of agents as someone or something that acts on one's behalf, but that seems to cover all of computers and software. Other than such generalities, there has been no consensus on the essential nature of agents." [14]

Different companies (IBM, Reticular Systems Inc, Crystaliz Inc, etc), research groups (MIT’s Media Lab) and individuals (Hayes-Roth, Woolbridge, Jennings, Brustolini, Maes) involved in agent research have offered a variety of definitions each hoping to explicate his or her use of the word “agent”. Perhaps they created their definitions based on the set of agents they had in mind [15]. Some definitions are long and complicated others are less rigorous. Some require only the capability for autonomous operations [15], while others insist on the capability for perceiving and affecting the environment [19].

This confusion can be explained by the inter-disciplinary character of agents, which on the one hand are subject to the effects of different scientific research directions, and on the other hand, reflect the requirements demanded in particular applications [16]. This
manuscript does not try to list all existing definitions of an agent (those interested in them may refer to [15][20]) because it could easily take several pages. For the purpose of this discussion we will define a software agent as a software component that [21]:

- executes autonomously,
- communicates with other agents or the user,
- monitors the state of its execution environment.

The main idea underlying software agents is that of delegation. The owner or user of a software agent delegates a task. It means that an intelligent agent performs a set of operations on behalf of a user who delegates the task to the agent [22]. The agent must be able to communicate with the user to receive its instructions and provide the user with the results of its activities. One may require an agent to possess high level of intelligence, but “intelligent agent software should not be viewed as being “smart” or “dumb”, but rather, should be viewed as having intellectual capabilities lying along a continuum” [21]. Agents that have more intelligence will have greater capabilities, but in certain applications software agents with limited capabilities may be required (Table 3.1).

Table 3.1: Attributes of an Intelligent Agent

<table>
<thead>
<tr>
<th>Agent</th>
<th>Executes autonomously</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Communicates with other agents or the user</td>
</tr>
<tr>
<td></td>
<td>Monitors the state of its execution environment</td>
</tr>
<tr>
<td>Intelligent Agent</td>
<td>Able to use symbols and abstractions</td>
</tr>
<tr>
<td></td>
<td>Able to exploit significant amounts of domain knowledge</td>
</tr>
<tr>
<td></td>
<td>Capable of adaptive goal-oriented behavior</td>
</tr>
<tr>
<td>Truly Intelligent Agent</td>
<td>Able to learn from the environment</td>
</tr>
<tr>
<td></td>
<td>Tolerant of errorful, unexpected, or wrong input</td>
</tr>
<tr>
<td></td>
<td>Capable of operation in real-time</td>
</tr>
<tr>
<td></td>
<td>Able to communicate using natural language</td>
</tr>
</tbody>
</table>
3.2 Classification

At the highest level, three major categories of agents can be distinguished: human agents, hardware agents, and software agents [16] (Figure 3.2).

![Figure 3.2 – Categories of intelligent agents.](image)

A situation with software agent classification is similar to one with agent definition – there are probably as many ways of classifying agent software as there are researches in the field. As Nwana [23] states, there are several dimensions to classify existing software agents: mobility (static and mobile agents), internal model (deliberate and reactive), autonomy, learning capability, cooperation, etc (Figure 3.3).

![Figure 3.3 – A part view of an agent topology](image)
The following table shows seven main types of agents and their short descriptions [23] (Table 3.2).

<table>
<thead>
<tr>
<th>1. Collaborative Agents</th>
<th>Emphasize autonomy and cooperation with other agents in order to perform tasks for their owners in multi-agent environment.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Interface Agents</td>
<td>Emphasize autonomy and learning in order to perform tasks for their owners.</td>
</tr>
<tr>
<td>3. Mobile Agents</td>
<td>Capable of roaming wide area networks such as WWW, interacting with foreign hosts, performing tasks on behalf of their owners and returning 'home'.</td>
</tr>
<tr>
<td>4. Information/Internet Agents</td>
<td>Manage, manipulate or collate information from many distributed sources.</td>
</tr>
<tr>
<td>5. Reactive Agents</td>
<td>Respond in a stimulus-response manner to the present state of the environment in which they are embedded.</td>
</tr>
<tr>
<td>6. Hybrid Agents</td>
<td>Combine several models within a single agent.</td>
</tr>
<tr>
<td>7. Smart Agents</td>
<td>Emphasize autonomy, cooperation, and learning capability.</td>
</tr>
</tbody>
</table>

Table 3.2: Types of agents

3.3 How information agents work

Each type of software agent has its own architecture. To save space and be more related to the subject of the thesis only one architecture will be discussed here – architecture of information agents (those interested in architectures of other types of agents can refer to [16][23]), because PaTra, an agent developed as a part of this thesis, is a framework for an information agent.
There is no standard mode of operation for information agents, for they may be static or mobile, non-cooperative or social, and may or may not learn [23]. Figure 3.4 (adapted from [23]) shows how a typical static information agent works.

An information agent within a browser uses internet management tools such as Spiders and search engines to gather information. The information agent may be associated with a spider. A spider is an indexer able to search the World Wide Web (WWW), depth-first, and store the topology of the WWW in a database management system (DBMS) and the full index of URLs in the WAIS (Wide Area Information Server).

3.4 Applications

Agent technology can be effectively applied in many areas including personal use, network management, information and Internet access, mobility management, e-commerce, computer user interface, application development, and military applications.
Currently many agent-based products are commercially offered in many areas. A number of vendors such as IBM (TIME 10 modules), Hewlett Packard (IT/O Agent Openview), BMC Group (Patrol Agent), and Computer Associates (TNG Agents) have developed network agents for corporate networks. The agent products for shopping on the Internet are Jango, Roboshopper, BargainFinder, BottomDollar, etc [22].

The market for agent software is rapidly expanding, and both vendor and client interest is beginning to pick up. A number of agents for different application areas are currently being developed in a variety of companies: AgentSoft, Botspot Inc, British Telecom, DEC, FTP Software Inc, IBM and many others.

3.5 Tools

An agent language is a programming or development language that can be used in the practical realization of intelligent software agents. Such a language should meet a number of major requirements [16]:

- **Object-orientedness.** Because agents are objects themselves, an agent language should support the object-oriented model.

- **Platform independence.** Since agents are used within different hardware and software environments.

- **Communications capability.** A must-have for multi-agent systems.

- **Security.** Mobile agents must especially have a high degree of security.
• **Code manipulation.** Many applications require that the program code of an agent is manipulated at runtime.

Table 3.3 adapted from [24] overviews agent languages presenting language categories and examples.

Table 3.3: Overview of agent languages

<table>
<thead>
<tr>
<th>Agent category</th>
<th>Language category</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooperation languages</td>
<td>Actor languages</td>
<td>Actors</td>
</tr>
<tr>
<td></td>
<td>Agent-oriented languages</td>
<td>Agent-0, Placa</td>
</tr>
<tr>
<td>Interface agents</td>
<td>Script languages</td>
<td>Tcl/Tk, Safe-Tcl</td>
</tr>
<tr>
<td>Information agents</td>
<td></td>
<td>Agent-Tcl, Java</td>
</tr>
<tr>
<td>Mobile agents</td>
<td></td>
<td>Telescript</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Active web tools</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Obliq</td>
</tr>
<tr>
<td></td>
<td></td>
<td>April</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scheme-48</td>
</tr>
<tr>
<td>Reactive agents</td>
<td>Reactive languages</td>
<td>RTA/ABLE</td>
</tr>
</tbody>
</table>

This table presents only some languages that could be used for agent programming. The number of agent development tools both commercial products (see Appendix A) and research projects (see Appendix B) is rapidly growing.
Chapter 4

PATRA (PATCH TRACKER)

4.1 Delegating patch tracking to an agent

The subject of this thesis - using an intelligent agent for software maintenance - was brought to life by an idea that some time-consuming software maintenance activities could effectively be delegated to a software agent. These activities, mentioned in the introduction, include:

- Keeping track of new patches,
- Determining what patches have to be applied to the system,
- Downloading appropriate patches,
- Applying patches.

The agent developed for this thesis – PaTra – is capable of keeping track of patches, so that a system administrator could delegate this task to the agent. Let’s look at this maintenance activity in more details.

First of all, a system administrator (for the purpose of the discussion this is the person responsible to keep track of patches) has to clearly understand the structure of a maintained system: what platforms are used, how many hosts, what software products are
installed and what their versions on each particular host are. Figure 4.1 illustrates an example of Oracle ERP system configuration for maintenance purposes.

Figure 4.1 – An example of Oracle ERP system configuration
This software system has four hosts: the development host is used to develop custom applications, the test host is used for testing developed applications, the training host is used to train employees for the new system, and the production host provides a production environment. The system is based on two different platforms – AIX and Solaris. Each host has a number of instances. Each instance on this figure has two major software components - DB (Oracle Database) and APPS (Oracle Applications) - and their versions. Each instance also can have other software products or modules installed – Human Recourses, Financials, Manufacturing and Distribution, Process Manufacturing, and others. Knowing the structure of the system, types of platforms being used, software products installed and their versions, a system administrator does a routine job – checks if there are any new patches for any components of the system available on the vendor’s web site. Oracle has a support web site called Metalink where all patches can be browsed by categories. By specifying a platform and a software product name on a search form (Figure 4.2), the system administrator gets a list of all patches for a particular software product (Figure 4.3). The next step is to manually identify if any new patches appeared on the list since the last revision. If any new patches are in the list, the system administrator should make a decision based on provided readme files whether these new patches should be applied. The process continues until all combinations of ‘Platform-Software Product’ presented in the system have been checked for new patches. Taking into account possible variety of platforms and installed software products, one can see that the process of keeping track of patches can be tedious.
This time-consuming procedure can be delegated to a software agent – PaTra. Let’s look at how PaTra simplifies the process of keeping track of patches. Once installed, the agent has to be configured before it can start doing the job. Information about the software system has to be added to the agent’s configuration file: all hosts of the system and all instances that reside on these hosts. Also PatchSets and PatchSetArrays should be defined. A PatchSet is basically a pair of values – Platform and Software Product – that is what a patch search form on Metalink requires as input. It can
be represented as a function $P(p,s)$ that has parameters $p$ (Platform) and $s$ (Software Product) as the input and a set of patches as the output. Some instances may require more than one PatchSet, that is why each instance is associated with a PatchSetArray—a set of PatchSets.

Figure 4.3 – A list of patches on Metalink

Figure 4.4 shows a generic configuration hierarchy of PaTra with $n$ hosts, each host has a number of instances and each instance has a number of PatchSets forming one
Once configuration is completed, the agent is ready to take over the job of patch tracking. PaTra is run on a regular basis, for example every night. The Cron utility on Unix can be used to schedule agent invocations. Upon invocation, the agent queries Metalink with every PatchSet it has in its configuration file and stores returned lists of patches into its database. So, if for example there are 50 PatchSets defined in the configuration file, PaTra will send 50 requests to Metalink and will get 50 lists of patches as the result. It is not difficult to imagine how much time it would take for the system administrator to do all this manually. Having completed querying Metalink, PaTra analyzes the patches it has just received from the web by comparing them against patches that had been in the database before the execution - this is how the agent detects new
patches. If new patches have been detected, PaTra sends an e-mail notification (that can be disabled) to the system administrator and terminates. If new patches have not been found, no notification is sent and the agent just terminates (Figure 4.5).

Figure 4.5 – The execution cycle of PaTra
Upon receiving the notification, the system administrator goes to PaTra's web interface (Figure 4.6) to take a look at the new patches and make a final decision whether these new patches have to be applied to the system or not. Patches are grouped by instances, which is a logical way to do, because DBAs apply patches on per-instance basis. PaTra presents patches sorted by their Status. New patches go first, then patches that have already been applied. Patches that are not applicable to the system are at the bottom of the list. The color of a patch on the list reflects the status of the patch: New patches are pink, Applied are gray, and Not Applicable are yellow. All possible statuses of patches are defined in the configuration file along with their colors and priorities they will have in the list. The system administrator can perform several operations with a patch on the list presented by PaTra. A patch can be deleted; the status of a patch and a note (just a text message) can be changed. An Update button at the top and the bottom of the web page commits changes.

PaTra is not just a notification tool that alerts the system administrator when new patches are released, but is also a configuration management tool that keeps track of all the patches that have been applied to the system. Because all patches are already in the agent's database along with their descriptions, there is no need for the system administrator to keep records of patches that have been applied (very often it is just an excel spreadsheet), which can save a lot of time.

Thus, PaTra can be effectively used as a tool for automating some software maintenance activities, saving the time of software maintenance personnel and therefore cutting the cost of software maintenance in general.
Figure 4.6 – PaTra’s Web interface
4.2 Design of PaTra

The design of PaTra has evolved from a rather simple one-file program that did not provide much functionality and flexibility to a somewhat complex software agent based on a three-tiered architecture that provides a high level of functionality and flexibility.

4.2.1 Three-tiered architecture

A three-tiered architecture became a choice for the design because it provides a high level of flexibility and extensibility. The tiers are somewhat independent from each other and it is possible to modify any of the tiers without any impact on the others. PaTra’s design consists of the following three tiers (Figure 4.7):

- **Knowbot tier** provides web-parsing capabilities,
- **Database tier** is an information storage of the agent,
- **Interface tier** provides communication with a system administrator.

The database tier connects the knowbot tier and the interface tier. The knowbot tier and the interface tier do not have any direct interactions. Moreover, these two tiers are even implemented on two different servers. The knowbot tier runs on IBM AIX, whereas the interface tier is located on Windows 2000 Professional.
4.2.2 The Knowbot Tier

A knowbot by definition is a program that automatically searches Internet sites and gathers information from them according to user-specified criteria. A knowbot is often called an intelligent agent or an agent. The term knowbot reflects the nature of PaTra's web-parsing module very well. The knowbot (see Appendix C) is written in Java-based scripting language WebL [25] that will be discussed later in this chapter.
4.2.3 The Database Tier

The database tier is PaTra’s information storage. It also provides analyzing capabilities, because it contains a PL/SQL package [26][27] with AnalyzePatches function (see Appendix D). The database module consists of seven tables:

- **PT_HOSTS** – contains hosts information;
- **PT_INSTANCES** – contains instances information;
- **PT_PATCHES** – holds all patches;
- **PT_RETRIEVED_PATCHES** – a temporary place to store retrieved patches. Once retrieved, patches in this table are compared against patches in the **PT_PATCHES** table to locate new ones and, after that the new patches are transferred to the **PT_PATCHES**;
- **PT_PATCHSETS** – stores all Patchsets;
- **PT_PATCHSETS_ARRAY** – contains all PatchSetArrays;
- **PT_PATCHES_STATUS** – has all defined statuses a patch can have;

4.2.4 The Interface Tier

The Interface tier (see Appendix E) provides communication capabilities for PaTra to interact with a system administrator (Figure 4.6). This Java Servlet module [28] communicates with the database tier by means of JDBC (Java DataBase Connectivity) [29].
4.3 Implementation of PaTra

The implementation phase of PaTra had several challenges. One was the user authentication and the other was a design of a simple, but powerful web parser. Oracle’s Metalink allows access only for authorized users – Oracle’s customers. Obviously, in order to query Metalink for patches PaTra has to be able to perform user authentication without user intervention.

The first language, rather an agent development environment, selected for PaTra’s implementation was AgentBuilder by Reticular Systems Inc. [21]. It turned out that this environment did not provide any effective ways for user authentication and web parsing. This was a reason to start searching for a language that would be more Web-oriented and would have the capabilities that AgentBuilder lacked. The second language of the choice was NQL (Network Query Language). NQL is a powerful scripting language ideal for building intelligent agents, bots and web applications. Strong communications is an important part of NQL’s feature set. Internet access to common protocols such as HTTP, FTP, NNTP, and TELNET are built into the language, as well as support for up-and-coming standards such as LDAP. E-mail is easily accessed, as are databases and desktop applications. Unfortunately, this language is a commercial product and its price was rather high. So, the search had to go on. Soon, the third and final selection was made – WebL – a programming language for the web.
4.3.1 WebL as a tool to build the knowbot module

WebL is a free, but yet powerful web scripting language for processing documents on the World Wide Web. It is well suited for retrieving documents from the web, extracting information from the retrieved documents, and manipulating the contents of documents. In contrast to other general purpose programming languages, WebL is specifically designed for automating tasks on the web. WebL language has a built-in knowledge of web protocols like HTTP and FTP. It also knows how to process documents in plain text, HTML and XML format [25]. WebL is written entirely in Java, which is both an advantage and a drawback. Advantages of being a Java product are:

- very easy to add bridges from WebL to Java code,
- Java objects can be called directly from WebL,
- Java extensions are loaded dynamically and it is possible to add and remove built-in functions by editing a standard script [25].

The challenges of the user authentication and web parsing were effectively overcome with this language. For example, this is a part of PaTra’s code responsible for the user authentication:

```plaintext
// Metalink supports Basic Access Authentication Scheme [30].

var A = "Basic " + Base64_Encode(LOGIN+"":"+PASSWORD);

// A single statement authenticates and retrieves the whole web page.

P = GetURL(URL2, [..], [ . Authorization = A .]);
```

That is all that takes to accomplish something that other languages have problems with.
Another excerpt from PaTra's code illustrates the ease of the web page parsing using WebL (Figure 4.8). This code actually retrieves all the attributes of a patch.

```javascript
var tables = Elem(P,"table")
directlyinside Elem(P,"table")[1];
every table in tables do
  // Get ID
  ID=Str Trim(Text(Elem(table,"td")[0]));
  // Get Patch name
  PatchName=Str Trim(Text(Elem(table,"td")[1],"a")[0]));
  // Get Readme file location
  Readme=Str Trim(Elem(Elem(table,"td")[2],"a")[0].href);
  // Get Product
  Product=Str Trim(Text((Elem(Elem(P,"table")
  after table)[0]));
  // Get Last Updated Attribute
  LastUpdated=Str Trim(Text((Elem(Elem(P,"table")
  after table)[1]));
  // Get Platform
  Platform=Str Trim(Text((Elem(Elem(P,"table")
  after table)[2]));
  // Get Patch Version
  PatchVersion=Str Trim(Text((Elem(Elem(P,"table")
  after table)[3]));
  // Get Info
  Info=Str Trim(Text((Elem(Elem(P,"table")
  after table)[4]));
  // Get Size of the Patch
  PatchSize=Str Trim(Text((Elem(Elem(P,"table")
  after table)[5]));
...
// End of the loop
end;
```

Figure 4.8 – Using WebL for web page parsing

It seems the only drawback that the language has is that WebL programs tend to run slowly sometimes, and their memory usage is quite high because WebL keeps everything in memory, including complete copies of pages. In PaTra's case it is not a problem, because real-time parsing of a web page is not required and the knowbot is run at night. Thus, WebL has proved to be a right choice for development of a software agent like PaTra.
4.3.2 Java Servlet for the User Interface

During the design stage, Perl was being considered as a tool to implement the user interface, but the final decision was Java Servlet. There are a number of advantages of using Java Servlet rather than CGI or Perl [28]:

- Servlets are portable, because they are written in Java;
- Servlet invocations are very efficient, because once loaded, a servlet usually remains in the server’s memory as a single object instance. In case of CGI or Perl a new process has to be spawn to handle a request;
- Servlets can handle errors safely due to Java’s exception handling mechanism;
- Servlets are object-oriented, which makes them very easy to develop;
- Servlets are better integrated with the server.

4.4 Result Validation

To validate the results of PaTra implementation, several people at the Ohio University Enterprise Project have tested the product in action (see Appendix F). A software validation questionnaire (see Appendix G) consisting of eight questions was developed to ask three people of their opinions. The following summarizes the results of the questionnaire.

In the question of whether such maintenance activities as finding new patches and “keeping track of patches” are routine and time-consuming, all answers were positive,
supporting the main goal of the PaTra project, which was automation of maintenance activities.

In the question of whether it was a good idea to automate these maintenance activities, everybody participated in the questionnaire agreed that these maintenance activities should be automated to a degree possible. This automation would help the application administrator in keeping systems up to date.

In the question of whether PaTra saves system administrator’s time, everybody agreed that PaTra does save system administrator’s time.

In the question of the positive features of PaTra, the answers could be grouped as follows:

- Powerful search and notification capabilities,
- Intuitive user interface,
- The framework of PaTra can easily be adapted and extended in other areas.

In the question of the negative features of PaTra, the main “negative” feature was described as being somewhat dependent on the specific host web site. If that site changes significantly, the application would likely need to change.

In the question “How would you extend this product?” questionnaires, participants would make PaTra more adaptable to changes in a software supplier’s web site design.

In the question of the usability of PaTra, all the participants agreed that PaTra provides a definite advantage to manually tracking down patches as well as an intuitive interface to log and track the status of such patches for multiple environments.
In the question of whether the use of PaTra increases security risks, all the answers were negative. The use of PaTra does not increase existing security risks. It only automates tasks that are already being performed by a system administrator. PaTra does not automatically download patches, but only their descriptions along with links to download web pages. This minimizes possible security risks.

4.5 Security Implications of using PaTra

In order to assess possible security implications related to PaTra, it is important, first of all, to analyze security risks that are present in the existing system and related to the following activities:

- Keeping track of new patches,
- Downloading patches,
- Applying patches.

At the Ohio University Enterprise Project, these procedures are performed manually by a database administrator. In order to find new patches, one has to use the Internet to log on to a vendor’s support site (Oracle’s Metalink) and browse through a list of newly released patches. The only security mechanism that Oracle provides to its clients is identification and authentication, which is done by means of a logon process. However, the lack of secure Internet connection (e.g. HTTPS) may easily undermine Oracle’s intention to identify and authenticate users of its support site. Because Internet connection is not secure, user ids and passwords being transmitted on a network can be
captured and used to gain access to legitimate users’ accounts. Patches being transmitted on a network can be maliciously modified in such a way that, once installed, they can harm or compromise legitimate users’ systems.

Having found a new patch that should be applied, a database administrator downloads it from the vendor’s support site. It would be a good practice for Oracle to provide some kind of digital signature along with patches to verify their origin and content. The lack of digital signatures in patches may theoretically jeopardize security of Oracle’s client’s systems, because as mentioned above, patches being insecurely transmitted over a network can be modified and a digital signature is a good verification mechanism that detects any illegitimate modifications of data.

PaTra aims to automate the first of the maintenance activities mentioned above – it keeps track of patches. PaTra does it in the same manner a database administrator does – by retrieving web pages with descriptions of patches from vendor’s support site. PaTra does not automatically download patches, but only their descriptions along with links to download web pages. Therefore, the use of PaTra does not increase existing security risks. It only automates tasks that are already being performed by a system administrator.
Chapter 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The goal of this thesis was to develop and implement a prototype of a software agent for helping in the maintenance process of a complex software system. The motivation for the tool was the abundance of routine work that is currently performed by system administrators responsible for software maintenance.

It was found that a number of researchers propose agent-based products for software maintenance, but none of the proposed systems are very practical and cannot be used for ERP and other complex software systems maintenance. The implementation and preliminary testing of PaTra has proved that a software agent can effectively be used for complex software systems maintenance.

PaTra has two main functions. It serves as a notification tool that alerts the system administrator when new patches are released - it visits the vendor’s support site on a regular basis to check if new patches are available. The agent is also a configuration management tool that keeps track of all the patches that have been applied to the system.

This thesis also proposed and implemented a three-tiered architecture in agent-based software systems, which helps in achieving flexibility and extensibility.
5.2 Future Work

PaTra has a great potential for improvement, because it automates only one of the four software maintenance activities performed on software consumer’s site – it can keep track of patches. The next maintenance activity, determining what patches have to be applied to the system based on patch descriptions, can be automated by adding some intelligence to the agent. This task would be much easier to solve with Oracle’s collaboration - if Oracle, for example, started to use structured or XML-based patch description. Another promising improvement would be to make PaTra highly adaptable. The current implementation of PaTra heavily relies on its configuration file, and in case of some changes in the maintained system (for example – a new host or instance is added) the configuration file has to be changed accordingly. It would be a great improvement for PaTra to be able to reconfigure itself in response to changing configuration of the system, although that could be a rather challenging and big project.

At the phase of designing PaTra a number of alternative architectures for the software agent were considered. One of them seems to be promising. It involves two software agents – one resides on the software producer’s site, and the other is on a customer’s site. The agents communicate or negotiate with each other. The customer’s agent informs the vendor’s agent about the configuration of the system and the vendor’s agent provides its counterpart with a precise list of patches that need to be applied to that system.
As one can see there are many directions to go from here and this is just another sign that the Intelligent Agent Technology in general and its applications in software systems maintenance specifically have a very good potential for growth.
REFERENCES


### APPENDIX A: Agent Construction Tools: Commercial Products

<table>
<thead>
<tr>
<th>Product</th>
<th>Company, remote link</th>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgenTalk</td>
<td>NTT and Ishida <a href="http://www.kecl.ntt.co.jp/csl/msrg/topics/at">http://www.kecl.ntt.co.jp/csl/msrg/topics/at</a></td>
<td>LISP</td>
<td>Multi-agent Coordination Protocols</td>
</tr>
<tr>
<td>Aglets</td>
<td>IBM Japan <a href="http://www.trl.ibm.com/aglets">http://www.trl.ibm.com/aglets</a></td>
<td>Java</td>
<td>Mobile Agents</td>
</tr>
<tr>
<td>Concordia</td>
<td>Mitsubishi Electric, Information Technology Center America <a href="http://www.merl.com/HSL/Projects/Concordia">http://www.merl.com/HSL/Projects/Concordia</a></td>
<td>Java</td>
<td>Mobile Agents</td>
</tr>
<tr>
<td>Gossip</td>
<td>Tryllian <a href="http://www.tryllian.com">http://www.tryllian.com</a></td>
<td>Java</td>
<td>Mobile Agents</td>
</tr>
<tr>
<td>Grasshopper</td>
<td>IKV++ <a href="http://www.grasshopper.de/index.html">http://www.grasshopper.de/index.html</a></td>
<td>Java</td>
<td>Mobile Agents</td>
</tr>
<tr>
<td>iGEN</td>
<td>CHI Systems <a href="http://www.cognitiveagent.com">http://www.cognitiveagent.com</a></td>
<td>C/C++</td>
<td>Cognitive Agent Toolkit</td>
</tr>
<tr>
<td>Intelligent</td>
<td>Bits &amp; Pixels <a href="http://www.bitpix.com">http://www.bitpix.com</a></td>
<td>Java</td>
<td>Agent Development Tool</td>
</tr>
<tr>
<td>Agent Factory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intelligent</td>
<td>Bits &amp; Pixels <a href="http://www.bitpix.com">http://www.bitpix.com</a></td>
<td>Java</td>
<td>Agent Library</td>
</tr>
<tr>
<td>Agent Library</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intelligent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agents</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product</td>
<td>Company/Institution</td>
<td>Programming Language</td>
<td>Category</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>----------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>Jumping Beans</td>
<td>Ad Astra Engineering, Inc.</td>
<td>Java</td>
<td>Mobile Components</td>
</tr>
<tr>
<td>LPA Agent Toolkit</td>
<td>Logic Programming Associates Ltd.</td>
<td></td>
<td>Agent Toolkit</td>
</tr>
<tr>
<td>MadKit</td>
<td>Madkit Development Group</td>
<td>Java, Scheme, Jess</td>
<td>Multiagent Development Tool</td>
</tr>
<tr>
<td>Microsoft Agent</td>
<td>Microsoft Corporation</td>
<td>Active X</td>
<td>Interface Creatures</td>
</tr>
<tr>
<td>NetStepper</td>
<td>JW's Software Gems</td>
<td></td>
<td>Agent Development Environment</td>
</tr>
<tr>
<td>Network Query Language</td>
<td>NQL Solutions</td>
<td></td>
<td>Programming Language</td>
</tr>
<tr>
<td>OpenCybele</td>
<td>Intelligent Automation Incorporated, MD</td>
<td>Java</td>
<td>Agent Infrastructure</td>
</tr>
<tr>
<td>Pathwalker</td>
<td>Fujitsu</td>
<td>Java</td>
<td>Agent-oriented Programming Library</td>
</tr>
<tr>
<td>RemoteBot.net</td>
<td>K-Team and Dreamlab</td>
<td>Java</td>
<td>Remote Robotics</td>
</tr>
<tr>
<td>Swarm</td>
<td>Swarm Development Group</td>
<td>Objective C, Java</td>
<td>Multiagent Simulation</td>
</tr>
<tr>
<td>Topia Personal Agents</td>
<td>Topia Ventures</td>
<td></td>
<td>Mobile Agents</td>
</tr>
<tr>
<td>UMPRS</td>
<td>Intelligent Reasoning Systems</td>
<td>C++</td>
<td>Agent Architecture</td>
</tr>
<tr>
<td>Voyager</td>
<td>Object Space</td>
<td>Java</td>
<td>Agent-Enhanced ORB</td>
</tr>
</tbody>
</table>
## APPENDIX B: Agent Construction Tools: Research Projects

<table>
<thead>
<tr>
<th>Product</th>
<th>Research Organization and remote links</th>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent Building Shell (ABS)</td>
<td>Enterprise Integration Laboratory, University of Toronto <a href="http://www.eil.utoronto.ca/ABS-page/ABS-overview.html">http://www.eil.utoronto.ca/ABS-page/ABS-overview.html</a></td>
<td>COOrderination Language (COOL)</td>
<td>Agent Architecture</td>
</tr>
<tr>
<td>Agent Factory</td>
<td>University College Dublin, Ireland <a href="http://krytan.ucd.ie/home.htm">http://krytan.ucd.ie/home.htm</a></td>
<td>VisualWorks</td>
<td>Agent Prototyping Environment</td>
</tr>
<tr>
<td>Agent Tcl</td>
<td>Dartmouth University <a href="http://agent.cs.dartmouth.edu/general/agenttcl.html">http://agent.cs.dartmouth.edu/general/agenttcl.html</a></td>
<td>Tcl</td>
<td>Mobile Agents</td>
</tr>
<tr>
<td>agentTool</td>
<td>Air Force Institute of Technology <a href="http://www.cis.ksu.edu/~sdeloach/ai/agentool.htm">http://www.cis.ksu.edu/~sdeloach/ai/agentool.htm</a></td>
<td>Java</td>
<td>Toolkit</td>
</tr>
<tr>
<td>Ascape</td>
<td>The Brookings Institution <a href="http://www.brook.edu/es/dynamics/models/ascape">http://www.brook.edu/es/dynamics/models/ascape</a></td>
<td>Java</td>
<td>Software Framework</td>
</tr>
<tr>
<td>Bee-gent</td>
<td>Toshiba Corporation, Systems and Software Research Laboratories <a href="http://www2.toshiba.co.jp/beegent/index.htm">http://www2.toshiba.co.jp/beegent/index.htm</a></td>
<td>Java</td>
<td>Software Development Framework</td>
</tr>
<tr>
<td>Bond Distributed Object System</td>
<td>Purdue University <a href="http://bond.cs.purdue.edu">http://bond.cs.purdue.edu</a></td>
<td>Java</td>
<td>Agent Framework</td>
</tr>
<tr>
<td>Cable</td>
<td>Logica Corporation <a href="http://public.logica.com/~grace/Architecture/Cable/public">http://public.logica.com/~grace/Architecture/Cable/public</a></td>
<td>Agent Definition Language, C++</td>
<td>System Architecture</td>
</tr>
<tr>
<td>D'Agents</td>
<td>Dartmouth University <a href="http://agent.cs.dartmouth.edu">http://agent.cs.dartmouth.edu</a></td>
<td></td>
<td>Retrieval, Organization, and</td>
</tr>
<tr>
<td>Tool</td>
<td>Institution</td>
<td>Language</td>
<td>Presentation</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DECAF Agent Framework</td>
<td>University of Delaware <a href="http://www.eecis.udel.edu/~decaf">http://www.eecis.udel.edu/~decaf</a></td>
<td>Java</td>
<td>Agent Framework</td>
</tr>
<tr>
<td>Gypsy</td>
<td>Technical University of Vienna <a href="http://www.infosys.tuwien.ac.at/Gypsy">http://www.infosys.tuwien.ac.at/Gypsy</a></td>
<td>Java</td>
<td>Mobile Agents</td>
</tr>
<tr>
<td>Hive</td>
<td>The Media Lab, Massachusetts Institute of Technology <a href="http://www.hivecell.net">http://www.hivecell.net</a></td>
<td>Java</td>
<td>Toolkit for Building Distributed Systems</td>
</tr>
<tr>
<td>Infospiders</td>
<td>University of California San Diego - Computer Science Dept <a href="http://dollar.biz.uiowa.edu/~fil/IS">http://dollar.biz.uiowa.edu/~fil/IS</a></td>
<td>Java</td>
<td>Adaptive Distributed Information Agents</td>
</tr>
<tr>
<td>JAFMAS</td>
<td>University of Cincinnati <a href="http://www.ececs.uc.edu/~abaker/JAFMAS">http://www.ececs.uc.edu/~abaker/JAFMAS</a></td>
<td>Java</td>
<td>Multiagent Framework</td>
</tr>
<tr>
<td>JATLite</td>
<td>Stanford University <a href="http://java.stanford.edu/java_agent/html">http://java.stanford.edu/java_agent/html</a></td>
<td>Java</td>
<td>Java Packages for Multiagents</td>
</tr>
<tr>
<td>JATLiteBean</td>
<td>University of Otago <a href="http://kmi.open.ac.uk/people/emanuela/JATLiteBean">http://kmi.open.ac.uk/people/emanuela/JATLiteBean</a></td>
<td>Java</td>
<td>JavaBean Component</td>
</tr>
<tr>
<td>KCLAIM</td>
<td>Universita' Di Firenze <a href="http://music.dsi.unifi.it">http://music.dsi.unifi.it</a></td>
<td>Klaim</td>
<td>Kernel Language for Agent Interaction and</td>
</tr>
<tr>
<td>Knowbot System Software</td>
<td>CNRI <a href="http://www.cnri.reston.va.us/home/ko">http://www.cnri.reston.va.us/home/ko</a> e</td>
<td>Python</td>
<td>Mobile Agents</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------</td>
<td>--------</td>
<td>---------------</td>
</tr>
<tr>
<td>Mobiware Middleware Tool</td>
<td>Columbia University <a href="http://comet.columbia.edu/mobiware">http://comet.columbia.edu/mobiware</a></td>
<td>Java</td>
<td>Mobile Networking Environment</td>
</tr>
<tr>
<td>MOLE</td>
<td>University of Stuttgart <a href="http://mole.informatik.uni-stuttgart.de">http://mole.informatik.uni-stuttgart.de</a></td>
<td>Java</td>
<td>Mobile Agents</td>
</tr>
<tr>
<td>Multi-Agent Modeling Language (MAML)</td>
<td>Central European University <a href="http://www.maml.hu">http://www.maml.hu</a></td>
<td>MAML</td>
<td>Programming Language</td>
</tr>
<tr>
<td>MultiAgent Systems Tool (MAST)</td>
<td>Technical University of Madrid <a href="http://www.gsi.dit.upm.es/~mast">http://www.gsi.dit.upm.es/~mast</a></td>
<td>C++</td>
<td>Multiple Heterogeneous Agents</td>
</tr>
<tr>
<td>NOMADS</td>
<td>Institute for Human and Machine Cognition, University of West Florida <a href="http://nomads.coginst.uwf.edu">http://nomads.coginst.uwf.edu</a></td>
<td>Java</td>
<td>Mobile Agents</td>
</tr>
<tr>
<td>RETSINA</td>
<td>Carnegie Mellon University <a href="http://www.cs.cmu.edu/~softagents">http://www.cs.cmu.edu/~softagents</a></td>
<td></td>
<td>Communicating Agents</td>
</tr>
<tr>
<td>Sensible Agents</td>
<td>The University of Texas at Austin, Laboratory for Intelligent Processes and Systems <a href="http://www.lips.utexas.edu/agents_on">http://www.lips.utexas.edu/agents_on</a> e_pager.htm</td>
<td>CORBA, Java, C, C++, Lisp</td>
<td>Autonomous Agent Specification and Architecture</td>
</tr>
<tr>
<td>Sodobot</td>
<td>MIT Artificial Intelligence Lab <a href="http://www.ai.mit.edu/people/sodabot/sodabot.html">http://www.ai.mit.edu/people/sodabot/sodabot.html</a></td>
<td></td>
<td>Software Agent User-environment and Construction System</td>
</tr>
<tr>
<td>SOMA (Secure and Open Mobile)</td>
<td>University of Bologna <a href="http://lia.deis.unibo.it/Research/SOM">http://lia.deis.unibo.it/Research/SOM</a> A</td>
<td>Java</td>
<td>Agent Programming Environment</td>
</tr>
<tr>
<td>Agent</td>
<td>Institution/Description</td>
<td>Language</td>
<td>Area</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------</td>
<td>----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>TeamBots</td>
<td>The Robotics Institute, Carnegie Mellon University <a href="http://www.teambots.org">http://www.teambots.org</a></td>
<td>Java</td>
<td>Multiagent Mobile Robotics</td>
</tr>
<tr>
<td>TuCSoN</td>
<td>Universita di Bologna <a href="http://www.lia.deis.unibo.it/Research/TuCSoN">http://www.lia.deis.unibo.it/Research/TuCSoN</a></td>
<td></td>
<td>Model For the Coordination of Internet Agents</td>
</tr>
</tbody>
</table>
APPENDIX C: Patra.Web1 – Implementation of a knowbot

// File Name: PATRA.WEBL
// Description: Retrieves Patches information from ORACLE's METALINK site
// Author: Mikhail P Dobrynin
// Created: 12/1/2001

Some Java classes used in this program need to be imported
import Str, Browser, Base64, Files;
import Java;

var CONFIGPATH = "\home/mikhail/patra/";
var CONFIG = CONFIGPATH + "patra.conf";

// Creating instances of Java Classes that will be used in the WebL script
var C = Java.Class("oracle.jdbc.driver.OracleDriver");
var System = Java.Class("java.lang.System");
var M = Java.Class("java.sql.DriverManager");
var Con = Java.Class("java.sql.Connection");
var St = Java.Class("java.sql.Statement");
var CST = Java.Class("java.sql.CallableStatement");
var St2 = Java.Class("java.sql.Statement");
var Res = Java.Class("java.sql.ResultSet");

Con=M.getConnection("jdbc:oracle:thin:login/password@hostname.ohiou.edu:portnumber:instance");

St=Con.createStatement();
St2=Con.createStatement();

var ID;
var PatchName;
var Readme;
var Product;
var LastUpdated;
var Platform;
var PatchVersion, Info, PatchSize;
// Site Info
var URL = Str.Trim(Call("cat $CONFIG | grep URL1: | awk '{print $2}"")");
var URL2 = Str.Trim(Call("cat $CONFIG | grep URL2: | awk '{print $2}"")");
var LOGIN = Str.Trim(Call("cat $CONFIG | grep LOGIN: | awk '{print $2}"")");
var PASSWORD = Str.Trim(Call("cat $CONFIG | grep PASSWORD: | awk '{print $2}"")");
var A = "Basic " + Base64_Encode(LOGIN+""+PASSWORD);
var P;
var S;
var Patchset;
var Options;
var ProductID;
var PlatformID;
var count;

St2.executeQuery("Delete from conv.PT_RETRIEVED_PATCHES");

Res=St.executeQuery("Select * from conv.PT_PATCHSETS");

while(Res.next()) do
Patchset=Res.getString("PATCHSET_ID");

Print( "Patchset: " + Res.getString("PATCHSET") );

P = GetURL(URL2, 
[..], [. Authorization = A .]);

Options= Elem(P,"option") contain Pat(P, Res.getString("PLATFORM") );
if Size(Options) == 1 then
  PlatformID = Options[0].value;
  Print(" ( " + Res.getString("PLATFORM") + ":" + PlatformID + ":" + ");

Options= Elem(P,"option") contain Pat(P, Res.getString("PRODUCT") );
if Size(Options) == 1 then
  ProductID = Options[0].value;
  PrintLn(Res.getString("PRODUCT") + ":" + ProductID + ");

count=0;

P = GetURL(URL+"?product-id=" + ProductID + ":&platform_id=" + PlatformID, 
[..], [. Authorization = A .]);

var tables = Elem(P,"table") directlyinside Elem(P,"table")[1];
every table in tables do

// Get ID
ID=Str_Trim(Text(Elem(table,"td")[0]));
// Patch name
PatchName=Str_Trim(Text(Elem(Elem(table,"td")[1],"a")[0] ));
// Readme file location
Readme=Str_Trim(Elem(Elem(table,"td")[2],"a")[0].href );
// Product
Product=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[0] ));
// Last Updated
LastUpdated=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[1]));
// Platform
Platform=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[2]));
// Patch Version
PatchVersion=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[3]));
// Info
Info=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[4]));
// Size
PatchSize=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[5]));

S="" + Str_Replace(ID,"","",") +
    "","" + Str_Replace(PatchName,"","",") +
    "","" + Str_Replace(Readme,"","",") +
    "","" + Str_Replace(Product,"","",") +
    "","" + Str_Replace(LastUpdated,"","",") +
    "","" + Str_Replace(Platform,"","",") +
    "","" + Str_Replace(PatchVersion","",") +
    "","" + Str_Replace(Info,"",") +
    "","" + Str_Replace(PatchSize,"",") +
    "";

St2.executeQuery("Insert into conv.PT_RETRIEVED_PATCHES " +
"( Patchset_ID, Patch_ID, ID, PATCH, README, PRODUCT, LAST_UPDATED, PLATFORM, 
VERSION, INFO, PATCH_SIZE, CREATION_DATE) " +
" values ( " + Patchset + " ,PT_PATCH_ID_S.NextVal, " + S + " ,sysdate ) ");

count=count+1;
Print("*");
end; // Patches loop
PrintLn(" ");
PrintLn(ToString(count) + " patch(es) retrieved");

else
    PrintLn(" Cannot find " + Res.getString("PRODUCT") );
end; // if end;
else
    PrintLn(" Cannot find " + Res.getString("PLATFORM") );
end; // if end;

end; // Patchsets loop

var exitstatus;
PrintLn("Analyzing retrieved patches...");
CSt=Con.prepareCall("{Call patra.AnalyzePatches} ");
CSt.execute();
PrintLn("Successfull!");
create or replace package body Patra as
--
procedure AnalyzePatches;

procedure AnalyzePatches
is
begin
-- Clean up retrieved data
update conv.pt_retrieved_patches
      set
         ID = DECODE(INSTR(ID, 'ID:'), 1, SUBSTR(ID,4),ID),
         PRODUCT = DECODE(INSTR(Product, 'Product:'), 1, SUBSTR(Product,9),Product),
         LAST_UPDATED = DECODE(INSTR(Last_updated, 'Last updated:'), 1, 
                              SUBSTR(Last_updated,14),Last_updated),
         PLATFORM = DECODE(INSTR(Platform, 'Platform:'), 1, SUBSTR(Platform,10),Platform),
         VERSION = DECODE(INSTR(Version, 'Patch version:'), 1, 
                         SUBSTR(Version,15),Version),
         INFO = DECODE(INSTR(Info, 'Info:'), 1, SUBSTR(Info,6),Info),
         PATCH_SIZE = DECODE(INSTR(Patch_Size, 'Size:'), 1, SUBSTR(Patch_Size,6),Patch_Size);
-- Analyzing ...
-- Copy to PT_PATCHES table only new patches from PT_RETRIEVED PATCHES table
insert into conv.pt_patches pp
(   PATCHSET_ID,
    INSTANCE_ID,
    Patch_ID,
    ID,
    Patch,
    Readme,
    Product,
    Last_Updated,
    Platform,
    Status,
    Version,
    Info,
    Patch_Size,
    Creation_date,
    LINK,
    NOTE)
select
      rp.PATCHSET_ID,
      pi.instance_id,
      rp.Patch_ID,
      rp.ID,
      rp.Patch,
      rp.Readme,
      rp.Product,
      rp.Last_Updated,
from conv.pt_retrieved_patches rp
, conv.pt_instances pi
, conv.pt_patchsets_array pa
where rp.PATCHSET_ID = pa.PATCHSET_ID
and pa.INSTANCE_ID = pi.instance_id
and not exists
(select * from conv.pt_patches p
    where p.instance_id = pi.instance_id
    and rp.ID = p.ID
    and rp.Patch = p.Patch);

commit;
end AnalyzePatches;

--
end Patra;
/
show errors
public class patra extends HttpServlet {
  private static String url = "jdbc:oracle:thin:login/password@hostname.ohiou.edu:port:instance";
  private static String NewStr = "New";
  HashMap Status = new HashMap();

  private static String AllHosts = "select ph.host-id, ph.host from conv.pthosts ph order by ph.HOST_id asc";

  public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    doGet(req, res);
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    Connection con = null;
    Statement stmt = null;
    Statement stmt2 = null;
    ResultSet rs = null;
    ResultSet rs2 = null;
    String host = ""; //"epprod";
    String inst = ""; //"eprd";
    String inst_id = "";
    String host_id = "";
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    try {
      // Load (and therefore register) the Oracle Driver
      // Database connection
      // SQL query to retrieve all host IDs and their corresponding host names
      // Process the query and display the results
      // Additional code for handling user interface
    }
  }
}
Class.forName("oracle.jdbc.driver.OracleDriver");

//DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

// Get a Connection to the database
con = DriverManager.getConnection(url);

// Create a Statement object
stmt = con.createStatement();

Enumeration enum = req.getParameterNames();
while (enum.hasMoreElements()) {
    String name = (String) enum.nextElement();
    if (name.compareToIgnoreCase("HOST") == 0) {
        if (name.equals("HOST1") == false) {
            host = req.getParameterValues(name)[0];
        }
    }
    if (name.compareToIgnoreCase("INSTANCE") == 0) {
        if (name.equals("INSTANCE1") == false) {
            instance = req.getParameterValues(name)[0];
        }
    }
    if (name.compareToIgnoreCase("HOST_ID") == 0) {
        if (name.equals("HOST_ID1") == false) {
            host_id = req.getParameterValues(name)[0];
        }
    }
    if (name.compareToIgnoreCase("INSTANCE_ID") == 0) {
        if (name.equals("INSTANCE_ID1") == false) {
            instance_id = req.getParameterValues(name)[0];
        }
    }
}

enum = req.getParameterNames();
int Updated, Deleted, flag;
Updated = 0;
Deleted = 0;
String PatchID = "", PatchID2 = "";

while (enum.hasMoreElements()) {
    String name = (String) enum.nextElement();
    String values[] = req.getParameterValues(name);
    if (values != null) {
        for (int i = 0; i < values.length; i++) {
            if (!name.endsWith(values[i])) {
                if (name.startsWith("status")) {
                    out.println("Patch-ID=");
                    PatchID = name.substring(name.indexOf('_') + 1,
                    name.lastIndexOf('-')) + 1, name.lastIndexOf('_'));
                    // Make changes in the database
                    out.println("<br>update conv.pt_patches pt set pt.status = " + values[i] + " where pt.patch_id = " +
                    stmt.executeUpdate("update conv.pt_patches pt set pt.status = " + values[i] + " where pt.patch_id = " +
                    name.substring(name.indexOf('_') + 1, name.lastIndexOf('_')) + 1, name.lastIndexOf('_')) +" and pt.instance_id = " + inst_id);
                    Updated++;
                }
            }
        }
    }
}

if (name.startsWith("note")) {
    out.println("Patch-ID=");
    PatchID2 = name.substring(name.indexOf('-') + 1,
    name.lastIndexOf('-')) + 1, name.lastIndexOf('-') + " where pt.patch_id = " +
    stmt.executeUpdate("update conv.pt_patches pt set pt.note = " + values[i] + " where pt.patch_id = " +
    name.substring(name.indexOf('_') + 1, name.lastIndexOf('_')) + 1, name.lastIndexOf('_')) +" and pt.instance_id = " + inst_id);
name.substring(name.indexOf('_')+l, name.lastIndexOf('-I)) + " and pt.instance_id = " + inst_id);
if ( ! PatchID2.equals(PatchID) )
    { Updated=Updated; }
}

if ( name.startsWith("check") ){

stmt.executeUpdate("delete from conv.pt_patches pt where pt.id = " +
name.substring(name.indexOf('_')+l, name.length()) + " and pt.instance_id = " + inst_id);

Deleted++;
while(rs.next()) {
    if (rs.getString("host").compareToIgnoreCase(host) == 0) {
        host_id = rs.getString("host-id");
        out.println("<font color="#FF0000">" +
            rs.getString("host").toUpperCase() + "<b>"
        );
        out.println("</font>" );
    } else
        out.println("&nbsp;<a HREF="http://" +
            hostname + "/se~let/patra?host=
            + rs.getString("host").toUpperCase()+"">" +
            rs.getString("host").toUpperCase() + "</a>" );
}

// Instances

if (host.equals("")) {
    out.println("<tr><td><b>Select a host</td><td>");
} else {
    rs = stmt.executeQuery("select pi.instance, pi.instance_id "+
        " from conv.pt_instances pi "+
        " where pi.HOST_ID = " + host_id);
    out.println("<tr><td><b>INSTANCES:</td><td>");
    while(rs.next()) {
        out.println("&nbsp;" );
        if (rs.getString("instance").compareToIgnoreCase(inst) == 0) {
            inst_id = rs.getString("instance_id");
            out.println("<font color="#FF0000">" );
            out.println("" + rs.getString("instance").toUpperCase()+" ");
            out.println("</font>" );
        } else
            out.println("<a HREF="http://" +
                hostname + "/servlet/patra?host="
                + host + "+instance=" + rs.getString("instance").toUpperCase() + "+" "+
                rs.getString("instance").toUpperCase()+"</a>" );
    }
}

out.println("</nobr></td></b></font>~itle~");
if (inst.equals("")) {
    // No Instance is defined
    out.println("<b>No instances defined for this host."");
} else
    out.println("<b>No instances defined for this host.");
}
out.println("<br><input type=hidden name=hostid value=" + host_id + ">");
out.println("<br><input type=hidden name=instance_id value=" + inst_id + ">");

// Print list of patches

// Populate Status HashMap
rs2 = stmt.executeQuery("select * from conv.pt_patches_status order by priority");
// rs2 = stmt.executeQuery("select * from conv.pt_patches");
int count=0;
while(rs2.next()) {
    Status.put(rs.getString("Status"), rs.getString("Color"));
    // out.println(rs.getString("Status") + " : " + rs.getString("Color") + " : " + rs.getString("Priority") + "
    count++;
}

String[] Statuses = new String[count];
Status.keySet().toArray(Statuses);
//
//        out.println(" Size: " + Statuses.length );
if (Deleted == 1)
{
    out.println("<br><b>" + Deleted + " Patch has been deleted");
}
else
    if (Deleted > 1)
    {
        out.println("<br><b>" + Deleted + " Patches have been deleted");
    }

    if (Updated == 1)
    {
        out.println("<br><b>" + Updated + " Patch has been updated");
    }
else
    if (Updated > 1)
    {
        out.println("<br><b>" + Updated + " Patches have been updated");
    }

rs = stmt.executeQuery("select count(*) COUNTER from conv.pt_patches pp where pp.instance_id = " + inst_id");
rs.next();

if (rs.getInt("COUNTER") == 0)
{
    out.println("<b>No patches found");
}
else{
// Print Buttons
out.println("<hr ALIGN="CENTER"/>");
out.println("<center><input type="submit" VALUE="Update" />
" + rs.getInt("COUNTER") + ")" );
out.println("<hr ALIGN="CENTER"/>");
out.println("&nbsp&nbsp&nbsp Status: &nbsp&nbsp <SELECT NAME="status-");
for (int i = 0; i < Statuses.length; i++) {
    out.print("<OPTION>");
    if ( Statuses[i].equals(rs.getString("Status") ) ) {
        out.print("SELECTED ");
    }
    out.println("">" + Statuses[i] );
}
out.println("</SELECT>");
out.println("&nbsp&nbsp&nbsp ID:" + rs.getString("ID") + "+" + "a HREF=" http://metalink.oracle.com/metalink/plsql/dis_downloadconfirm?p_id=" + rs.getString("ID") + "+" + rs.getString("PATCH") + "</a>");
out.println("" + rs.getString("README") + "">README.txt</a></small>");
out.println("<img src="http://" + hostname + "/images/delete.gif" align="MIDDLE">" +
"<input TYPE="CHECKBOX" NAME="check_" + rs.getString("ID") + ";" +
"</small><"+<"tr">" +
out.println("<td><small>Product:" +
rs.getString("PRODUCT") +
"</small></td>" +
out.println("<td><small>Last updated:" +
rs.getString("Last_Updated") +
"</small></td>" +
out.println("<td><small>Platform:" +
rs.getString("PLATFORM") +
"</small></td>" +
out.println("<td><small>Patch version:" +
rs.getString("PLATFORM") +
"</small></td>" +
out.println("<td><small>Info:" +
rs.getString("INFO") +
"</small></td>" +
out.println("<td><small>Size:" +
rs.getString("Patch_Size") +
"</small>";" +
out.println("<td><br/>
<\font>";
}
out.println("<\br></td></\b></font></\b></table >";
out.println("<\TD></\TR><\TD BGCOLOR="\#FFFFFF" COLSPAN=3">";
out.println("<hr ALIGN="\"CENTER\"">";
out.println("<\center<input type="\"submit" VALUE="\"Update\" >";
out.println("<hr ALIGN="\"CENTER\"">";
out.println("</TD></\TR>";
out.println("<\tr<\td ALIGN="\"LEFT\" BGCOLOR="\#FFFFFF" VALIGN="\"BOTTOM\"">";
out.println("<img SRC="\"http://" + hostname + "/images/kur_left_bottom.gif" ALIGN="\"ABSBOTTOM\"">";
*/
String Note = rs.getString("Note");
if ( Note.equals("null") )
{
Note="";
}
rs2 = stmt.executeQuery("select * from conv pt_instances pi where pi.HOST_ID = " + rs.getString("host_id")");
while(rs2.next()) {
    out.println("<LI>" + rs.getString("instance") );
}
/*
 */
out.println("</FORM>");
out.println("</BODY></HTML>");
}
}
catch(ClassNotFoundException e) {
    out.println("Couldn't load database driver: " + e.getMessage());
}
catch(SQLException e) {
    out.println("SQLException caught: " + e.getMessage());
}
finally {
    // Always close the database connection.
    try {
        if (con != null) con.close();
    }
    catch (SQLException ignored) { }
}
APPENDIX F: Demonstrating the operation of PaTra

PaTra has two functions. It is a notification tool that alerts a system administrator when new patches are released. PaTra is also a configuration management tool that keeps track of all the patches that have been applied to the system.

PaTra’s knowbot module runs on a regular basis (nightly). Figure F.1 is a sample run of PaTra’s knowbot module.

![A sample run of PaTra's knowbot module](image)

Figure F.1 – A sample run of PaTra’s knowbot module

First the knowbot loads a system configuration from PaTra’s configuration file patra.conf. In the sample configuration file four hosts were defined: Development (instances: EDEV, ECNV), Test (instances: ETST, EOBJ), Production (instances: EPRD,
EDBA), and Training (instances: VIS, EEOY, ESSA, EGLD). Also, four patch sets were defined in the configuration file: Payroll, Alert, Application Object Library, Alert (different version), and Human Resources. Having loaded the system configuration, the knowbot queries Metalink’s patch search form for every patch set defined in patra.conf. Each star symbol in the output represents one patch retrieved. PaTra does not download patches themselves – only their descriptions. Upon retrieving all patch descriptions for a patch set, the knowbot outputs a number of patches retrieved for this patch set. Once all patch sets have been processed, PaTra executes PL/SQL procedure that “analyzes” retrieved patch descriptions to detect new patches. If there are new patches, the knowbot sends a notification e-mail (Figure F.2) to an address specified in PaTra’s configuration file.

![PaTra has detected new patches](image)

PaTra has detected 7 new patch(es) on Metalink. Please visit PaTra’s web site to see new patch(es).

Figure F.2 – A notification e-mail sent by PaTra
Upon receiving the notification, the system administrator goes to PaTra’s web interface to review the new patches and make a final decision whether these new patches have to be applied to the system or not. Patches are grouped by instances (ECNV, EDEV) (Figure F.3), because DBAs (Database Administrators) apply patches on per-instance basis.

![PaTra interface](image)

**Figure F.3 – Patches are grouped by instances in PaTra**

PaTra presents patches sorted by their Status (Figure F.4). New patches go first followed by patches that have already been applied. Patches that are not applicable to the system are at the bottom of the list. The color of a patch on the list reflects the status of the patch: New patches are pink, Applied are gray, and Not Applicable are yellow. All
possible statuses of patches are defined in the configuration file along with their colors and the priorities they will have in the list. Along with a list of patches PaTra displays the total number of patches for an instance.

![PaTra's web interface](image)

**Figure F.4 – PaTra’s web interface**

A system administrator can change the status of a patch (Figure F.5) by selecting a different status in a pull down list of values and then clicking Update button to commit.
changes. A patch can be deleted by checking a check box next to a stop sign. To commit the deletions the commit button has to be pressed. A system administrator can also enter a description in the Note field of a patch. This description can be used to store any comments related to a patch.

Figure F.5 – Changing a patch status and deleting patches using PaTra’s web interface
Upon committing changes PaTra confirms deletions and updates (Figure F.6).

Figure F.6 – PaTra’s confirmation of patch deletions and updates

PaTra can output patches based on their status: New, Applied, or Not Applicable.

To view patches with a particular status, a status check box should be selected (Figure F.7).
Figure F.7 – PaTra presents only those patches whose status is checked

The number of patches per page can be changed by entering a desired number in a Patches-Per-Page text field and clicking the Update button (Figure F.8). If there are more patches than can be fit on a page, page numbers are displayed above the list of patches. Patches for a particular page can be viewed by clicking on the page number. This ability
is especially useful when there are a lot of patches and it could be slow to put them all on one page.

Figure F.8 – Number of patches per page can be adjusted
By clicking on the Readme link of a patch a system administrator can go directly to Metalink’s Readme file of the selected patch (Figure F.9).

Figure F.9 – A Readme file is retrieved directly from Metalink
A system administrator can download a patch by selecting a Patch number, which is a link to Metalink’s patch download web page where the selected patch can be downloaded directly from Oracle’s site (Figure F.10).

![A patch is downloaded directly from Metalink](image)

Figure F.10 – A patch is downloaded directly from Metalink

This concludes the sample run of PaTra.
APPENDIX G: Software Validation Questionnaire for PaTra

1. Do you think such maintenance activities as finding new patches and "keeping track of patches" are routine and time-consuming?

2. Is it a good idea to automate these maintenance activities? If not – explain why.

3. Do you think PaTra saves system administrator’s time?

4. What are positive features of PaTra?

5. What are negative features of PaTra?

6. How would you extend this product?

7. Would you use this product? If not - explain why.

8. Do you think the use of PaTra increases security risks?