Using Agents to Reduce Maintenance Cost of Complex Software Systems

COSTAS VASSILIADIS and MIKHAIL DOBRYNIN
School of Electrical Engineering and Computer Science
Ohio University, Main Campus
Athens, Ohio 45701
USA

Abstract: - PaTra (PAtch TRAcker) is an agent capable of keeping track of patches that have been applied (old) and have to be applied (new) to a complex software system. Patra uses Oracle's Metalink, a support web site that can provide a list of patches for a specific product and platform. The agent queries daily, to get a list of patches for platforms and products that are specified in its configuration file. PaTra finds new patches, stores their information into its database and sends electronic notifications to a system administrator. Upon receiving the notifications, the system administrator uses PaTra's interface to examine the new patches and make a final decision whether they have to be applied to the system or not. Although in its current implementation PaTra can be used with Oracle's ERP system, it can be modified for other complex software systems with a similar scheme of interaction between a software producer and a software consumer. This is a time consuming process especially when there are several instances of the same system and 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 work proposes to delegate the task of keeping track of patches to an intelligent software agent. This automation saves time and cuts the cost of maintaining the system in general.

Key-Words: - Intelligent Agents, Software Maintenance, ERP Systems, Web-based Applications

1 Introduction

Approximately 70 percent of the total cost of owning software is related to maintenance. Both software designers and software users of large complex systems deal with the maintenance of the systems. Designers have to support the systems they sell by creating patches and upgrades to the various modules while consumers are responsible for keeping the systems up and running.

Companies that have deployed such systems always assign teams responsible for the maintenance of the system. The list of their activities usually include: a) Keeping track of new patches, which come out on a regular basis and are usually accessible through the vendor's web site. For example Oracle has a tech-support web site called Metalink, where all the patches can be browsed by categories, and where patches are listed with descriptions and prerequisites, b) Determining which patches have to be applied to the specific system. While some patches have prerequisites, and others are cumulative, any complex software system keeps an internal track of the patches that have been applied, c) Downloading appropriate patches. With an FTP session established, and user authentication required, patches are downloaded to a local host, d) Applying patches. This process is complicated and difficult to generalize. There are software systems such as the Kintana system, that have been specifically developed for patch application. Kintana keeps track of the patches that have been applied to the system. It defines a workflow for each patch application. This is needed as application process may consist of several steps which have to be approved by different offices.

All of the operations above are time consuming. Software systems based on intelligent agent technology can effectively be used to automate some of these procedures and significantly reduce their cost. 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 twenty thousand times per year. One of the steps taken by many software companies is using the Internet as a tool to interact with customers by means of a support web site. On these sites users can create requests for assistance. On Oracle's support site 'Metalink', these requests are called TARs – Technical Assistance Requests, Customers search for new patches, download them,

and have access to extensive documentation libraries and knowledge bases. This increases the quality of customer support and lowers the cost of software maintenance.

Keeping track of patches can effectively be automated by using intelligent agent technology. 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 intelligence to the agent and with 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 very difficult 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 was solved, then it will be 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 application is established.

2 Problem Formulation

It is important to make a distinction between an upgrade and a patch. Although these terms appear to be similar they have significant differences. An upgrade implies that a version of a software system changes. Upgrades replace old systems with a newer one. A patch, on the other hand, is the immediate solution to a problem. It can sometimes be downloaded from the vendor's Web site. The patch is not necessarily the best solution for the problem and the product developers often find a better solution and they offer it with the product's next release. The application of patches only partially changes a software system, whose version usually stays the same.

Patches are usually used when the cost associated with the development and the application of the patch is significantly less than the cost associated with the development and the application of an upgrade. That is why simple software systems almost never have patches, while users constantly upgrade them. On the other hand, complex software systems usually get upgraded once every few years, while being constantly patched. Because large number of patches in a software system may contribute to instability and security problems, software developers try not to issue too many patches between software upgrades.

2.1 Trends in Software Upgrading

The Internet offers new interaction possibilities between software producers and consumers. Many software companies already have on-line services to support their products. For example, 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. Symantec's LiveUpdate makes it easy to keep Norton Utilities up-to-date with the latest program updates.

Many software products currently 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 simultaneously searches for installed versions of software applications and hardware drivers. The results page shows the components that CatchUp did locate on the PC that are supported by the service. Then a user can select software components to be updated and the CatchUp software downloads the selected patches and updates via the Internet and installs them on a PC.

Thus, 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 research efforts on dynamic software updating [1] and Configurable Distributed Systems [2] 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.

2.2 Delegating Patch Tracking to an Agent

PaTra, as an agent is capable of keeping track of patches, which allows the system administrator to delegate this task to the agent. The system administrator should first be able to understand the structure of the maintained system: what platforms are used, how many hosts, what software products are installed and what their versions on each particular host are. PaTra was developed for Oracle's ERP system. This 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 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 their system, system administrators routinely check if there are any new patches for any components of the system available on the vendor's web site. By specifying a platform and a software product name on a search form, the system administrator gets a list of all patches for a particular software product.

The next step is to manually identify if any new patches appeared on the list since the last revision, in which case a decision has to be made based on the information provided in the readme files. This process continues until all combinations of 'Platform-Software Product' presented in the system have been checked for new patches.

This time-consuming procedure is delegated to PaTra. Once installed, information about the software system has to be added to the agent's configuration file. Also PatchSets and PatchSetArrays should be defined. A PatchSet is basically a pair of values – Platform and Software Product – required by the patch search form on Metalink. The set is 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 an instance is associated with a PatchSetArray.

Once configuration is completed, the agent is ready for patch tracking. PaTra is run 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. Having completed querying Metalink, PaTra analyzes the patches it has just received by comparing them against patches that had been in the database before the execution. If new patches are detected, PaTra sends an electronic notification to the system administrator and terminates. Upon receiving the notification, the system administrator goes to PaTra's web interface to examine whether these new patches need to be applied to the system or not. Patches are grouped by instances. PaTra presents patches sorted by their Status. 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 reflects its status. 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. PaTra is not just a notification tool that alerts the system administrator when new patches are released, it 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, a time consuming process.

3 System Design

PaTra is a complex software agent based on a three-tiered architecture (Fig. 1), that provides a high level of functionality and flexibility. The tiers are somewhat independent from each other and it is possible to modify any of the tiers without any impact on the others. The design consists of the **Knowbot tier** written in WebL [3] which provides web-parsing capabilities, the **Database tier** which is an information storage for the agent, and the **Interface tier** which provides communication with the 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.

The Interface tier provides communication capabilities for PaTra to interact with a system administrator. A Java Servlet module [4] communicates with the database tier by means of JDBC [5].

3.1 Implementation Issues

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

The development environment, selected for PaTra's implementation was AgentBuilder by Reticular Systems Inc. [6]. It turned out that this environment did not provide any effective means for user authentication and web parsing. The second 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. The third and final selection was WebL, a programming language for the web.

WebL is a free, powerful web scripting language (Fig. 2), for processing documents on the web. It is well suited for retrieving documents, extracting information from them, and manipulating their contents. In contrast to other general purpose programming languages, WebL is specifically designed for automating tasks on the web. WebL 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. WebL is written entirely in Java.

The challenges of the user authentication and web parsing were effectively overcome with this language. The only drawback was 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.

3.2 Security Implications with PaTra.

In order to assess possible security implications related to PaTra, it is important to analyze security risks that are present in the existing system, related to the following activities: Keeping track of new patches, Downloading patches, Applying patches.

In the current system, these procedures are performed manually by a database administrator. In order to find new patches, one has to use the Internet to log on 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 a secure Internet connection 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. 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.

PaTra's automation mimics the job of database administrator in retrieving web pages with descriptions of patches from vendor's support site. PaTra does not automatically download patches, but only their descriptions along with web links for downloading. Therefore, the use of PaTra does not increase existing security risks. It only automates tasks that are already being performed by a system administrator.

4 Conclusions

The goal of this work 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 reduction of routine work that is currently performed by system administrators responsible for software maintenance.

While a number of researchers have proposed agent-based products for software maintenance, none of them are very practical and they cannot be applied to ERP and other complex software systems. The implementation and preliminary testing of PaTra has proven that a software agent can effectively be used for complex software systems maintenance. PaTra was built in a three-tiered agent-based architecture, which helped in achieving flexibility and extensibility. PaTra has two main functions. First, it serves as a notification tool that alerts system administrators when new patches are released - it visits the vendor's support site on a regular basis to check if new patches are available. Second, it is used as a configuration management tool that keeps track of all the patches that have been applied to the system.

PaTra has a great potential for improvement, as it automates only one of the four software maintenance activities performed on consumer sites

– the keeping 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 changes in the maintained system (for example – a new host or instance 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 changes in the configuration of the system, although that could be a rather challenging and large undertaking. During the phase of PATra's design 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 on the 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 large potential for growth.

References:

[1] Hicks, M. "Dynamic Software Updating." *SIGPLAN Conference on Programming Language Design and Implementation*, 2001.

[2] Scott, R.H. "Agent-based Software Configuration and Deployment.", *Ph.D. Dissertation, University of Colorado*, 1999.

[3] "Agent Builder." *Reticular Systems Inc*, 1999.

[4] Hannes, M. "WebL – a programming language for the Web." *Compaq Systems Research Center*, 1999.

[5] Hunter, J. Java Servlet Programming. *O'Reilly*, 1998.

[6] Hamilton, G., Cattel, R., Fisher, M. JDBC Database Access with Java. Addison-Wesley, 1997.

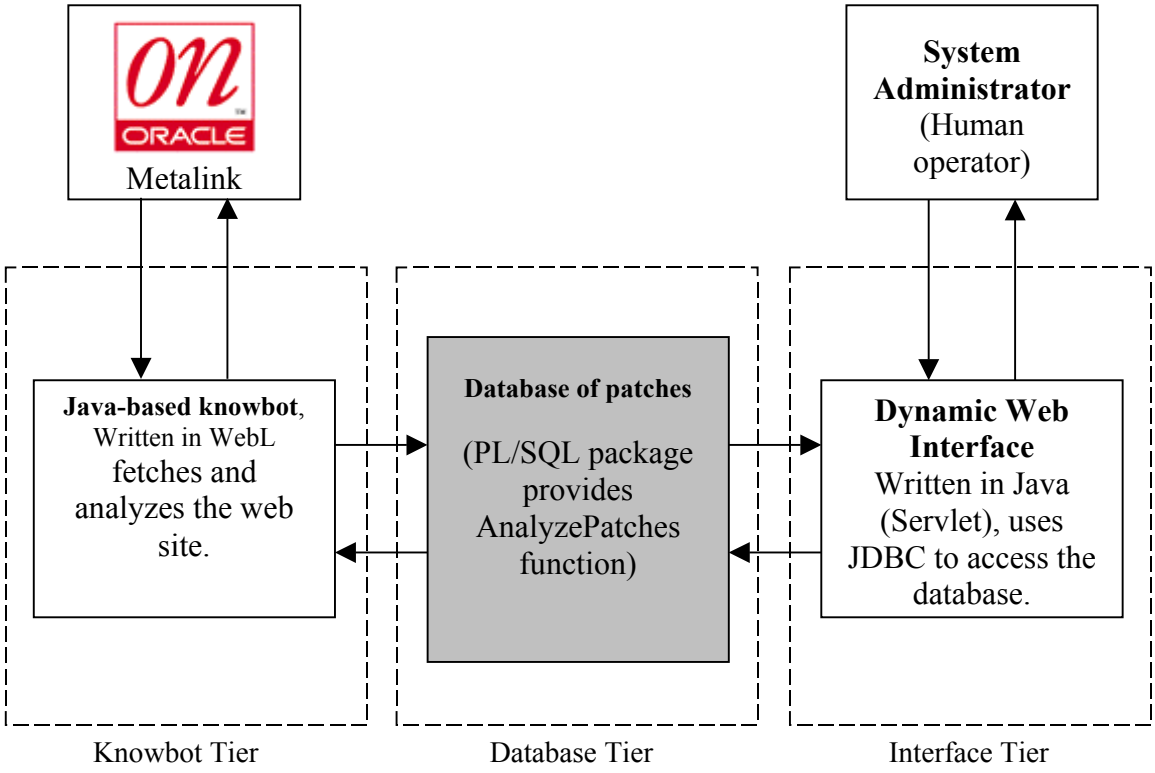

Fig. 1 Three-Tiered Architecture of Patra

```
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(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")[1],"td") after table)[0] ));
// Get Last Updated Attribute
LastUpdated=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[1] ));
// Get Platform
Platform=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[2] ));
// Get Patch Version
PatchVersion=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[3] ));
// Get Info
Info=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[4] ));
// Get Size of the Patch
PatchSize=Str_Trim(Text((Elem(Elem(P,"table")[1],"td") after table)[5] ));
……
// End of the loop
end;
```

Fig. 2 Using WebL for Web Page Parsing. Code retrieves all attributes of a patch.