International Journal of Engineering Inventions
e-ISSN: 2278-7461, p-ISSN: 2319-6491
Volume 4, Issue 2 (August 2014) PP: 50-57

Internet Security Agent

Inas Ismael Imran
Baghdad University-College of Education for Women-Computer Dept

Abstract: *Security has been identified as the major concern for the agent paradigm for two reasons. First, foreign code that executes on a site shares that site's services and resources with local processes and other agents. Services can include electronic commerce utilities. Resources include the file system, the GUI and the network server, as well as memory and CPU. It is difficult for a site to ensure that no agent can steal information or corrupt another agent or shared resource. The second security problem is that the agent itself can be circumvented by a malicious site which may steal or corrupt agent data or simply destroy the agent. To solve this problems we build a mini–password manager using a code in language Java. Then we incorporate the mini–password manager into the simple web server to authenticate users that would like to download documents and resources. The goal of this paper is to accentuate the positive aspects that agents bring to Internet security.*

I. Introduction

Internet security is a tree branch of computer security specifically related to the Internet, often involving browser security but also network security on a more general level as it applies to other applications or operating systems on a whole. Its objective is to establish rules and measures to use against attacks over the Internet.[1] The Internet represents an insecure channel for exchanging information leading to a high risk of intrusion or fraud, such as phishing.[2]. Today, the Internet is the computer. The Internet brings further problems for security. The openness of the network means that one can never identify all users who may need to gain access to a service.[3]

Agent are software entities that have enough autonomy and intelligence to carryout various tasks with little or no human intervention .they are software delegates of individuals and organization , and can act on behalf of their delegators[4]. In order to support systems, agent-based systems would have to satisfy the following requirements:[5]

- A popular, easy to use language for writing agents.
- Cross-platform support for agents.
- Support for agents that have not been certified or identified and authenticated.
- Secure execution environments for untrusted agent code.
- Support for agent mobility.
- Mechanisms for inter-agent communication.
- User interface mechanisms for communication between agents and users.
- Agent locater services.
- Access to Internet resources through standard Internet protocols and standards.

II. Objectives

- Discuss ways of making computer systems secure.
- Describe common threats that need to be addressed by web programmers.
- Describe the impact of malware on computer networks.
- Describe the following: Spyware, identity theft, fraud.

III. Overview

There are many risks that we face when connecting to the internet.

1. Hacking - where a cyber intruder will attempt to gain access to your computer to capture. passwords, credit card details, software.
2. Viruses - attacking your system via email attachments (macro virus, exe programs).
3. Buggy Beta software - as it is distributed via the net more likely you will be tempted to use Beta software.
4. Identity theft - taking of personal information and using it to commit a crime.
5. Fraud - criminal deception, use of false representations. E.g. Using Credit card numbers.
6. Cyber stalking - Where your movements are tracked. This can become very unnerving.
7. Invasion of privacy - As you use the web companies like DoubleClick.com may be collecting information about you.

IV. Picking A Security Policy

A security policy is the set of decisions that, collectively, determines an organization's posture toward security. More precisely, a security policy determines the limits of acceptable behavior, and what the response to violations should be. Naturally, security policies will differ from an organization to organization. But every organization should have one, if only to let it take action when unacceptable events occur.

The first step, then, is to decide what is and is not permitted. To some extent, this process is driven by the business or structural needs of the organization; thus, there might be an edict that bars personal use of corporate computers. Some companies wish to restrict outgoing traffic, to guard against employees exporting valuable data. Other aspects may be driven by technological considerations: a specific protocol, though undeniably useful, may not be used, because it cannot be administered securely. Still others are concerned about employees importing software without proper permission: the company doesn't want to be sued for infringing on someone else's rights. Making such decisions is clearly an iterative process, and one's answers should never be carved in stone or etched in to silicon.[6]

V. Security Requirements

The users of networked computer systems have four main security requirements: confidentiality, integrity, availability, and accountability.[7]

5.1. Confidentiality

Any private data stored on a platform or carried by an agent must remain confidential.Agent frameworks must be able to ensure that their intra- and inter-platform communications remain confidential. Eavesdroppers can gather information about an agent's activities not only from the content of the messages exchanged, but also from the message flow from one agent to another agent or agents. Monitoring the message flow may allow other agents to infer useful information without having access to the actual message content. A burst of messages from one agent to another, for example, may be an indication that an agent is in the market for a particular set of services offered by the other agent. The eavesdropping agent or external entity may use this information to gain an unfair advantage. Agents may be able to detect an Agent Communication Language (ACL) conversation signature pattern to infer further meaning from an agent conversation. A procurement agent may, for example, send three messages to a vendor agent, followed by one message to its home platform, and conclude the transaction with two messages to the vendor agent. The vendor agent may send two messages to a credit checking agency and conclude with a message to its banking agent. Although the contents of the messages have never been disclosed, an eavesdropper may be able to infer that the procurement and vendor agents have successfully completed a sale of some commodity or service.

5.2. Integrity

The agent platform must protect agents from unauthorized modification of their code, state, and data and ensure that only authorized agents or processes carry out any modification of shared data. The agent itself cannot prevent a malicious agent platform from tampering with its code, state, or data, but the agent can take measures to detect this tampering.

5.3. Accountability

Each process, human user, or agent on a given platform must be held accountable for their actions. In order to be held accountable each process, human user, or agent must be uniquely identified, authenticated, and audited. Example actions for which they must be held accountable include: access to an object, such as a file, or making administrative changes to a platform security mechanism. Accountability requires maintaining an audit log of security-relevant events that have occurred, and listing each event and the agent or process responsible for that event. Security-relevant events are defined in the platform security policy and should include, at a minimum, the following: user/agent name, time of event, type of event, and success or failure of the event. Audit logs must be protected from unauthorized access and modification. Measures need to be in place to prevent auditable events from being lost when the storage media reaches its capacity. These measures can include alerting the system administrator when the audit logs reach a certain capacity and depend on the system administrator to perform routine maintenance as part of their normal administrator duties.

5.4. Availability

The agent platform must be able to ensure the availability of both data and services to local and remote agents. The agent platform must be able to provide controlled concurrency, support for simultaneous access, deadlock management, and exclusive access as required. Shared data must be available in a usable form, capacity must be available to meet service needs, and provisions for the fair allocation of resources and timeliness of service must be made. The agent platform must be able to detect and recover from system software

and hardware failures. While the platform can provide some level of fault-tolerance and fault-recovery, agents may be required to assume responsibility for their own fault-recovery.

VI. Security Threats

Jackson (2009) identified the following threats that need to be considered by web programmers as:[8]

- **User input** - Don't trust user input. E.g. can add JavaScript into a comment box.
- **Phishing** - A fake version of your web site .
- **XSS** – Cross-site scripting.
- **CDRF** - Cross domain request forgery (Posting a form from one website to another) .
- **Click-jacking**. Example program at Guya.net.

VII. Security Malware

We describe types of malware such as rootkits, botnets, and keyloggers—and how these have posed threats to the security of the Internet and electronic commerce. Some of the malware that we have described take advantage of software vulnerabilities to spread, but other types of malware may not. Some malware, for instance, may rely on social engineering–based attacks to dupe users into downloading and installing it. Since 2003, the types of malware that we list have become more prevalent, and we provide some more up-to-date information and some case studies of them at www.learnsecurity.com/ntk.

Here are some types of malware that you need to be aware of:[9]

7.1. Worms

A worm is a type of a virus. A virus is a computer program that is capable of making copies of itself and inserting those copies into other programs. One way that viruses can do this is through a floppy or USB disk. For instance, if someone inserts a disk into a computer that is infected with a virus, that virus may copy itself into programs that are on the disk. Then, when that disk inserted in other computers, the virus may copy itself and infect the new computers. A worm is a virus that uses a network to copy itself onto other computers. The rate at which a traditional virus can spread is, to an extent, dependent upon how often humans put infected disks into computers. A worm, on the other hand, uses a network such as the Internet to spread. Millions of computers are always connected to the Internet. The rate at which a worm can propagate and spread to other computers is orders of magnitude faster than the rate at which viruses can spread for two reasons: (1) there are a large number of available computers to infect, and (2) the time required to connect to those computers is typically on the order of milliseconds.

7.2. Rootkits

A rootkit is a set of impostor operating system tools (tools that list the set of active processes, allow users to change passwords, etc.) that are meant to replace the standard version of those tools such that the activities of an attacker that has compromised the system can be hidden. Once a rootkit is successfully installed, the impostor version of the operating system tools becomes the default version. A system administrator may inadvertently use the impostor version of the tools and may be unable to see processes that the attacker is running, files or log entries that result from the attacker's activity, and even network connections to other machines created by the attacker.

7.3. Botnets

Once an attacker compromises ("owns") a machine, the attacker can add that machine to a larger network of compromised machines. A botnet is a network of software robots that attackers use to control large numbers of machines at once. A botnet of machines can be used, for example, to launch a DDoS attack in which each of the machines assimilated into the botnet is instructed to flood a particular victim with IP packets. If an attacker installs a rootkit on each machine in a botnet, the existence of the botnet could remain quite hidden until the time at which a significant attack is launched.

7.4. Spyware

Spyware is software that monitors the activity of a system and some or all of its users without their consent. For example, spyware may collect information about what web pages a user visits, what search queries a user enters, and what electronic commerce transactions a user conducts. Spyware may report such activity to an unauthorized party for marketing purposes or other financial gain.

7.5. Keyloggers

A keylogger is a type of spyware that monitors user keyboard or mouse input and reports some or all such activity to an adversary. Keyloggers are often used to steal usernames, passwords, credit card numbers, bank account numbers, and PINs.

7.6. Adware

Adware is software that shows advertisements to users potentially (but not necessarily) without their consent. In some cases, adware provides the user with the option of paying for software in exchange for not having to see ads.

7.7. Trojan horses

Also known simply as a Trojan, a Trojan horse is software that claims to perform one function but performs an additional or different function than advertised once installed. For example, a program that appears to be a game but really deletes a user's hard disk is an example of a Trojan.

7.8. Clickbots

A clickbot is a software robot that clicks on ads (issues HTTP requests for advertiser web pages) to help an attacker conduct click fraud. Some clickbots can be purchased, while others are malware that spreads like worms and are part of larger botnets. They can receive instructions from a botnet master server as to what ads to click, and how often and when to click them. Some clickbots are really just special cases of a bot in a botnet.

VIII. Agent Security Framework

In the past, as teams of individuals have developed agent systems, pragmatics prevailed and emphasis was placed on functionality over security. While some agent system implementations incorporate appropriate security techniques, often little regard is given to interoperability among agent systems. What is needed is an overall framework that integrates compatible techniques into an effective security model and provides an umbrella under which interoperability can exist.

The Foundation for Intelligent Physical Agents' (FIPA) '97 and '98 standards and Object Management Group's MASIF standards both fall short in providing the desired framework. The FIPA work is focused mainly on standardizing the agent communication language used among cooperating agents. Many of the details regarding the architecture of the agent platform require significant work before any substantive progress can be made on security. The MASIF standards on the other hand do make a clear and definitive statement on security, relying on the CORBA security services architecture.

Unfortunately, although the CORBA model adequately addresses security services for an agent platform, it largely ignores any independent security services needed by an agent.[7]

IX. The Agent Model: Places, Agents And Messages

The Internet application executes over a set of inter-connected sites or *places*. An agent is an autonomous program that accesses resources locally at a place; when it needs to use resources at another place, it migrates to that place [10]. Some agents at a place are system agents: these do not move and typically offer services such as storage, GUI, etc. [11]

A **says** m **to** B
A **moves to** P **as** B
A **Requests** C **from** P
new *A a*
start
Terminate A

The set of actions that an agent can execute are listed in this table. At each place, agents exchange messages with one another and with their environment through the **says** action where m is the message exchanged. A and B are agents, or one of them is the local environment, denoted Env. There are two base component types of messages: simple text strings and *Names*. A text string is not interpreted by the environment. For instance, a password exchanged between a visiting agent and its hosting site can be represented as a string or as (the name of) a mobile password object that contains the password string. A name denotes an agent or place and it is evaluated in the current environment. Thus the name *stdio* at place client can refer to a different agent to that named *stdio* at place server. A message can also be some sequential combination of names and strings, but nothing more.

Agents are mobile meaning that they can move or be moved between sites. This is done using the **moves** action where agent A moves to place P and is named B there. When a new agent is required by an agent at a site, it can ask for it with the **Request** action. This means that agent A requests that the agent named C

resident at place P be sent from place P to the current environment. This might be useful for instance to download a certificate (agent) if the executing agent is involved in some authentication step. Recall that a certificate not just be some static data structure, but a program that executes. The **Request** action of course can be modeled by a **move** and a **says** but it is included for conciseness in agent policies.

Two actions are supported by the architecture for agent startup. The first is **start** and is executed by an agent on its initialization. It is the signal to the agent that it has just been created. The **new** action creates an agent of a specified *kind* A that becomes named a. An agent's kind specifies the behavior of an agent in the same way that a class describes the behavior of an object. We will return to this feature in the subsection on security where an agent's kind is simply the set of policy rules defined for the agent. The final action of the agent architecture is **Terminate** A which kills an agent named A in the current environment. Only the environment can execute this command.

X. Protecting Agents

While countermeasures directed towards platform protection are a direct evolution of traditional mechanisms employed by trusted hosts, and emphasize active prevention measures, countermeasures directed toward agent protection tend more toward detection measures as a deterrent. This is due to the fact that an agent is completely susceptible to an agent platform and cannot prevent malicious behavior from occurring, but may be able to detect it.

The problem stems from the inability to effectively extend the trusted environment of an agent's home platform to other agent platforms. While a user may digitally sign an agent on its home platform before it moves onto a second platform, that protection is limited. The second platform receiving the agent can rely on the signature to verify the source and integrity of the agent's code, data, and state information provided that the private key of the user has not been compromised. On the agent's subsequent hop to a third platform, the initial signature from the first platform remains valid for the original code, data, and state information, but not for any state or data generated on the second platform.

For some applications, such minimal protection may be adequate. For example, agents that do not accumulate state or convey their results back to the home platform after each hop have less exposure to certain attacks. For other applications, simple schemes may prove adequate. For example, the Jumping Beans [12] agent system addresses some security issues by implementing a client-server architecture, whereby an agent always returns to a secure central host first before moving onto any other platform. Agent systems that allow for more decentralized mobility, such as IBM Aglets, prevent the receiving platform from accepting agents from any agent platform that is not defined as a trusted peer within the receiving platform's security policy. Alternatively, the originator can restrict an agent's itinerary to only a trusted set of platforms known in advance.

While these simple schemes have value, they do not support the loose roaming itineraries envisioned in many agent applications. Some more general-purpose techniques for protecting an agent include the following:

- Partial Result Encapsulation.
- Mutual Itinerary Recording.
- Itinerary Recording with Replication and Voting.
- Execution Tracing.
- Environmental Key Generation.
- Computing with Encrypted Functions.
- Obfuscated Code (Time Limited Blackbox).

XI. Safe Code Interpretation

Agent systems are often developed using an interpreted script or programming language. The main motivation for doing this is to support agent platforms on heterogeneous computer systems. Moreover, the higher conceptual level of abstraction provided by an interpretative environment can facilitate the development of the agent's code [13]. The idea behind Safe Code Interpretation is that commands considered harmful can be either made safe for or denied to an agent. For example, a good candidate for denial would be the command to execute an arbitrary string of data as a program segment.

One of the most widely used interpretative languages today is Java. The Java programming language and runtime environment [14] enforces security primarily through strong type safety. Java follows a so-called sandbox security model, used to isolate memory and method access, and maintain mutually exclusive execution domains. Security is enforced through a variety of mechanisms. Static type checking in the form of byte code verification is used to check the safety of downloaded code. Some dynamic checking is also performed during runtime. A distinct name space is maintained for untrusted downloaded code, and linking of references between modules in different name spaces is restricted to public methods. A security manager mediates all accesses to system resources, serving in effect as a reference monitor. In addition, Java inherently supports code mobility, dynamic code downloading, digitally signed code, remote method invocation, object serialization, platform

heterogeneity, and other features that make it an ideal foundation for agent development. There are many agent systems based on Java, including Aglets [15, 16], Mole [17], Ajanta [18], and Voyager [19].

XII. Practical Work

12.1. Good password procedure

1. Do not use your login name in any form(as is ,reversed , capitalized doubled, etc).
2. Do not use your first, middle, or last name in any form or use your children name.
3. Do not use other information easily obtained about you. this includes license plate numbers, telephone numbers, social security numbers, the name of street you live on, etc.
4. Do not use a password of all digits or all the same letters.
5. Do not use a word contained in English or foreign language dictionaries, spelling lists, or other lists of words.
6. Do not use a password shorter than six characters.
7. Do use a password with mixed-case alphabetic.
8. Do use a password with non alphabetic characters (digit or punctuation).
9. Do use a password that is easy to remember, so you don't have to write it down.

12.2. Algorithms

When a user tries to login, we could simply locate the corresponding username in the file, and do a string comparison to determine whether the password that the user enters matches the one in the password file. If the username does not appear in the password file, the login would, of course, be denied. In this paper we use three codes to check the password in java and we use anther code in JavaScript to check password strength . 1.The code checkPassword():- method looks up the username in the dUserMap hashtable and compares the password provided to the one stored in the hashtable. If they match, checkPassword() returns true; otherwise checkPassword() returns false, quite simple and straightforward.

```
public static boolean checkPassword(String username, String password) {
 try {
 HashedPasswordTuple t = (HashedPasswordTuple)dUserMap.get(username);
 return (t == null) ? false :
 t.getHashedPassword().equals(getSaltedHash(password, t.getSalt()));
      } catch (Exception e) {
      }
      return false;
  }
```

2.The code add():- method simply adds an entry to the dUserMap hashtable, which is keyed by username and stores the password as the value.

```
public static void add(String username, String password) throws exception
{
int salt = chooseNewSalt ();
HashedPasswordTuple ur=
new HashedPasswordTuple(getSaltedHash(password, salt), salt);
dUserMap.put(username,ur);     }
```

The code show If the password false alarm say their someone tries to input the system and close the communication.

```
 while ((line = br.readLine()) != null) {
 int delim = line.indexOf(DELIMITER_STR);
 String username=line. Substring(0,delim);
 HashedPasswordTuple ur =
 new HashedPasswordTuple(line.substring(delim+1));
 userMap.put(username, ur);
 }
} catch (Exception e) {
System.err.println ("Warning: their someone tries to input the system.");
 }
```

```
return userMap;
line. close()
}
```

3.The following code for add(),check password() and computeSHA()shows how to implement a mini –password manger that hashes passwords for brevity, it only shows those methods that need to be modified. Note that in preceding code, the computeSHA() method is called with the password as an argument in both the add() and check password() methods. The computeSHA() method uses a MessageDigest object provided as part of the java security package. Once an instance of MessageDigest object that can compute SHA-256 hashes is obtained, the MessegeDigest objects update() method is called with the bytes that make up the input string(the password). Then the hash is computed by calling the digest() method.the hashed bytes are then base 64 encoded to substitute nonprintable characters for printable ones, and the hash of input string is returned.

```
private static String computeSHA(String preimage) throws Exception {
MessageDigest md = null;
md = MessageDigest.getInstance("SHA-256”);
md.update(preimage.getBytes("UTF-8”));
byte raw[] = md.digest();
return (new sun.misc.BASE64Encoder().encode(raw));
```

4.Here is simple method to check the password strength using JavaScript, just copy and paste the script given below in your registration field and customize it accordingly.

```
<script language="javascript">
function passwordCheck()
{var strongRegex = new RegExp("^(?=.{8,})(?=.*[A-Z]) (?=.*[a-z])(?=.*[0-9])(?=.*\\W).*$", "g");
var mediumRegex = new RegExp("^(?=.{7,})(((?=.*[A-Z])(?=.*[a-z]))|((?=.*[A-Z])(?=.*[0-9]))|((?=.*[a-z])(?=.*[0-9]))).*$", "g");
var enoughRegex = new RegExp("(?=.{6,}).*", "g");
var pwd = document.getElementById("password");
if (pwd.value.length==0) {
document.getElementById('check').innerHTML = 'Type Password';
} else if (false == enoughRegex.test(pwd.value)) {
document.getElementById('check').innerHTML = 'More Characters';
} else if (strongRegex.test(pwd.value)) {
document.getElementById('check').innerHTML = '<b><span style="color:green">
Strong!</span>';
} else if (mediumRegex.test(pwd.value)) {
document.getElementById('check').innerHTML = '</b><b><span style="color:orange">
Medium!</span>';
} else {
document.getElementById('check').innerHTML='</b><b><span style="color:red">Weak!</span>';
}}
</b></script>
<input name="password" id="password"
type="text" size="15" maxlength="20"
onkeyup="return passwordCheck();" />
<span id="check">Type Password</span>
```

XIII. Conclusion

Any web sites, operating systems, and other types of software have been built to use passwords to authenticate users. Although the security community has been working over the years to move toward systems that use more sophisticated authentication mechanisms, it is likely that password systems will be in use for some time. Hence, it is important to understand the strengths and weaknesses of passwords systems, and how to make them less vulnerable to attacks.

In this paper, we have proposed four coded for password to protect system from any one try to enter it , and we are also interested in keeping the agent as intelligent and autonomous as possible, allowing it to use this code (even if very simple) when needed.

REFERENCE

[1]. Gralla, Preston , "How the Internet Works. Indianapolis", Pub. ISBN 0-7897-2132-5, 2007.
[2]. Rhee, M.Y., "Internet Security: Cryptographic Principles,Algorithms and Protocols", ISBN 0-470-85285-2, 2003. .
[3]. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. "InSympSecPr, Research in Security and Privacy", Oakland, CA, IEEE Computer Society,Technical Committee on Security and Privacy, IEEECSP, May 1996..
[4]. Jeff Rosenschien ,Tuomas Sandholm ,Carles Sierra, Pattie Maes ,and Rob Guttmann,"Agent-mediated electronic commerce : Issue ,challenge and some viewpoints", In Proceeding of the Second International Conference on Autonomous Agents, page 189-196, May 1998.
[5]. Niranjan Suri , Kenneth M. Ford, and Alberto J. Cafias," An Architecture for Smart Internet Agents, Institute for Huma&n MachineCognition" ,University of West Florida, nsuri@ai,uwf.Edu, 1998.
[6]. J.P. Holbrook, J.K. Reynolds, "Site Security Handbook". RFC 1244, Jul-01, 1991.
[7]. Wayne Jansen and Tom Karygiannis, "Mobile Agent Security-Computer Security" ,National Institute of Standards and Technology ,Special Publication 800-19, 44 pages, October 1999.
[8]. Jackson, K."Web Security" ,3 June , 2009 from http://pageofwords.com/blog/2009/05/28/WebSecurityNdashNapier.aspx
[9]. Neil Daswani, Christoph Kern, and Anita Kesavan," Foundations of Security: What Every Programmer Needs to Know", ISBN-13 (pbk): 978-1-59059-784-2 ,ISBN-10 (pbk): 1-59059-784-2, 2007.
[10]. J. Vitek and C. Tschudin. "Mobile Objects Systems". Springer Verlag, Berlin, 1997.
[11] J. H. Morin and D. Konstantas. Hypernews:" A Media application for the commercialization of an electronic newspaper". In Proceesings of SAC '98 - The 1998 ACM Symposium on Applied Computing, Marriott Marquis, Atlanta, Georgia, U.S.A, Feb. 27 - Mar. 1 1998.
[12]. Ad Astra Engineering Inc. ,"Jumping Beans White Paper, Sunnyvale CA, December 1998.
[13]. John K. Ousterhout, "Scripting: Higher-Level Programming for the 21 st Century,"IEEE Computer, pp. 23-30, March 1998.
[14]. A. Fuggetta, G.P. Picco, and G. Vigna, "Understanding Code Mobility," IEEE Transactions on Software Engineering, 24(5), May 1998. <URL: http://www.cs.ucsb.edu/~vigna/listpub.html>
[15]. James Gosling and Henry McGilton, "The Java Language Environment: A White Paper," Sun Microsystems, May 1996. <URL: http ://j ava. sun. cm/docs/white/langenv/>
[16]. Gunter Karjoth, Danny B. Lange, and Mitsuru Oshima, "A Security Model For Aglets," IEEE Internet Computing, pp. 68-77, August 1997.
[17] Markus StraBer, Joachim Baumann, Fritz Hohl, "Mole - A Java Based Mobile Agent System," in M. Miihlhauser (ed.), Special Issues in Object Oriented Programming, Verlag, pp. 301-308,1997.<URL:http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/ ECOOP96.ps.gz>
[18] Neeran Karnik, "Security in Mobile Agent Systems," Ph.D. Dissertation, Department of Computer Science, University of Minnesota, October 1998.
[19] ObjectSpace Inc.,"ObjectSpace Voyager Core Package Technical Overview," version 1.0, December 1997.<URL: http://www.objectspace.com/developers/voyager/white/index.html>