Enhancing Security Of Agent-Oriented Techniques Programs Code Using Jar Files

Vivek Jaglan

Research Scholar
Suresh Gyan Vihar University
Jaipur, India

Surjeet Dalal

Asst. Professor CSE Dept
Manav Rachna International University
Faridabad, India

Dr. S. Srinivasan

Professor Department of Computer Applications,
PDMCE Bahadurgarh
Jhajjar, India

Abstract— Agent-oriented techniques characterize an exciting new way of analyzing, designing and building complex software systems in real time world. These techniques have the prospective to significantly improve current practice in software engineering and to extend the range of applications that can feasibly be tackled. The Agent Oriented Programming & Software Engineering is being focused to design distributed software systems. The agent technology has been dominated to enhance software reusability and maintainability.

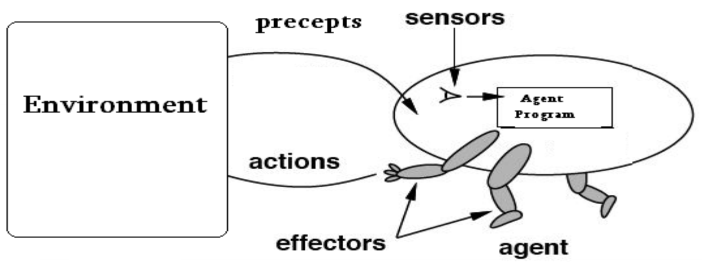
Now more than 100 Agent toolkits for commercial & research purpose are available. Most of the toolkits are designed using Java language. Although use of agent toolkits makes programming more efficient, but the security of code is not achieved. The security of the program code cannot be ignored. In this paper we discuss the approach of using jar file to enhance the security. The JAR files can be electronically signed & verified. The digital certificates can be used by the programmers. By using the jar file agent program code can be more secure. In this paper we will discuss an approach to achieve the security of the code of agent programs. Now we will discuss how jar file are created & security is to be implemented.

Keywords- Intelligent agent; Agent-oriented Programming; JAR File; Security;

I. INTRODUCTION

The vast growth in the field of software engineering over the past two decades has been made through the development of increasingly powerful and natural high-level abstractions with which leads to model and develop complex systems. The various components like procedural abstraction, abstract data types, and, most recently, objects are all examples of such abstractions. The agent-oriented technologies represent a similar advance in abstraction. These technologies may be used in field of software development for developing more logically & more efficient software system for complex distributed systems. If the software developers want to utilize full potential of agents, then the agent-oriented technologies should be considered as a software engineering paradigm. For this purpose it is necessary to develop software engineering techniques that are specifically tailored to agent-oriented technologies.

The intelligent agents may be defined as the software entity which is capable of interacting with the environment through sensor & acts on them through the effectors. The environments may be defined as the set of information & facts in the any system. For example, the environment of travelling system may be set of information regarding the fares, distance between two places, arrival & departure time.



When an agent perceives its environment, perhaps by means of a sensor of some kind, it is common to model that in terms of percepts. When an agent interacts with its environment in order to make a change in the environment, it is known as an action. The mechanism by which this is accomplished is often called an effector. Agent behaviour can be classified as reactive or proactive. A reactive agent only responds to its environment. These changes are communicated to the agent as events. Thus, changes in the environment have an immediate effect on the agent. The agent merely reacts to changing conditions and has no long-term objectives of itself. In contrast, a proactive agent has its own objectives.

The key characteristics of agents are widely understood to be highly autonomous, proactive, situated, and directed software entities. Other characteristics such as mobility are optional and create a special subtype of agent. The uniqueness of agents is said to be that they are proactive (as well as reactive), have a high degree of autonomy, and are positioned in and interact with their environment. For achieving a particular goal, it constructs a specified plan and then to execute that plan by means of process elements known as actions. In reality, many agents are designed as hybrid agents, possessing both reactive and proactive characteristics. The challenge then is for the designer to balance these two very different behaviours in order to create an overall optimal behaviour.

The behavior of the agent is defined by various intentional concepts such as goals, beliefs, abilities, commitments, etc., provide a higher-level characterization of behavior. One can characterize an agent in terms of its intentional properties without having to know its specific actions in terms of processes and steps. The unambiguous representation of goals allows motivations and rationales to be expressed. Beliefs provide for the possibility that an agent can be wrong in its assumptions about the world, and mechanisms to support revisions to those assumptions. The one well-known agent architecture that reflects many of these notions is the Beliefs, Desires, and Intentions (BDI) architecture. Beliefs, Desires, and Intentions are seen as high-level, abstract, externally ascribed characteristics. These three characteristics are then mapped through to the design or model layer. An appropriate concept of *agent* can be developed to serve as the central construct in a new kind of modeling and analysis to respond to today's needs. Much like the concepts of activity and object that have played pivotal roles in earlier modeling paradigms, the agent concept can be instrumental in bringing about a shift to a much richer, socially-oriented ontology that is needed to characterize and analyze today's systems and environments.

II. RELATED WORK

Many Papers have been published relating to Agent oriented programming. In 1991 Yoav Shoham presented new computational framework called agent-oriented programming (AOP), which can be viewed as a specialization of object-oriented programming. The state of an agent consists of components such as beliefs, decisions, capabilities, and obligations; for this reason the state of an agent is called its mental state. The mental state of agents is described formally in an extension of standard epistemic logics beside temporalizing the knowledge and belief operators, AOP introduces operators for obligation, decision, and capability. Agents are controlled by agent programs, which include primitives for communicating with other agents. In the spirit of speech act theory, each communication primitive is of a certain type: informing, requesting, and offering, and so on. This article presented the concept of AOP, discussed the concept of mental state and its formal underpinning, defined a class of agent interpreters, and then described in detail a specific interpreter that has been implemented. Babak Hodjat introduced an adaptive agent Oriented software architecture that is a new approach to software design based on an agent-oriented architecture is presented. Unlike current research, we consider software to be designed and implemented with this methodology in mind. In this approach agents are considered adaptively communicating concurrent modules which are divided into a white box module responsible for the communications and learning, and a black box which is the independent specialized processes of the agent. A distributed Learning policy is also introduced for adaptability.

In 2000, Nicholas R. Jennings and Michael Wooldridge presented Agent-oriented techniques as a new means of analyzing, designing and building complex software systems. In 2003 Sunil Doiphode showed how agent-oriented programming used in Defence Domain. This paper gave an overview of this paradigm, its benefits over the other conventional programming paradigms being used. It also proposes the decision support system model for the military domain. In the proposed system there are certain critical issues, which need to be focused upon.

The existing conventional paradigms are inadequate to deal with these issues. This paper identified these critical issues and discusses how AOP can address these issues.

In 2006 Louise A. Dennis introduced Common Semantic Basis for BDI Languages. He provided a common semantic basis for a number of BDI languages, clarifying issues and aiding further programming language development & supporting formal verification by developing a model-checker optimized for checking AIL programs – existing BDI languages can have compilers for AIL so as to take advantage of its associated model-checker; and providing, potentially, a high-level virtual machine for efficient and portable implementation.

In 2008 Albert Treytl introduced a secure agent platform for active RFID. RFID are used to identify products in factory automation applications; this article introduces a platform that allows to combine both technologies and to run a software agent on an active RFID device and as a result gain increased flexibility and control.

In 2009 Peter Novak discussed Code Patterns for Agent-Oriented Programming. He proposed a purely syntactic approach to designing an agent programming language. On the substrate of Behavioral State Machines (BSM), a generic modular programming language for hybrid agents, he showed how an agent designer can implement high-level agent-oriented constructs in the form of code patterns (macros). In this paper, he put forward an alternative approach to design of agent-oriented programming languages. On the level of a generic language for programming reactive systems, he proposed development of a library of code patterns (macros, templates) whose semantics refers to various agent oriented concepts, such as an achievement goal or a maintenance goal.

III. AGENT-ORIENTED PROGRAMIMG

The term AOP (Agent-oriented programming) was introduced by Shoham, in 1993. Agent oriented programming is promising as a new paradigm for constructing software systems. On the basis of the concept of software agent, a number of new kinds of systems are now being developed. According to one definition (Jennings et al., 1998), software agents are situated: they sense the environment and perform actions that change the environment; they are autonomous: they have control over their own actions and internal states, and can act without direct intervention from humans; and they are flexible: approachable to changes in the environment, goal-oriented, opportunistic, and take initiatives; they are also social - they work together with other artificial agents and humans to complete their tasks and help others. These conventional programming platforms like structural programming, object-oriented programming (OOP), logical programming, etc. were inadequate in some of the application areas.

The OOP approach has been used for developing intelligent agents, but there exists the main limitation of OOP is that it cannot represent complex mental attitudes. With logical programming, it is possible to represent and conclude relationships among mental attitudes, such as intentions, goals, and beliefs, with limitations in the usage of capabilities of action. Agent oriented programming is capable of designing the software system in such new way of thinking about what software is and how it should be constructed. In comparison of object oriented programming, agent oriented programming supports a higher level of abstraction for thinking about the characteristics and behaviors of software systems. It can be seen as part of an ongoing trend towards greater interactivity in conceptions of programming and software system construction.

While developing more powerful and robust software capabilities is clearly fundamental to the enterprise of software and information systems, it is also clear that we need effective techniques for determining the needs and requirements for particular application settings, so that the right system will be designed to fulfill those requirements. The success of any software system depends critically on the requirements and how well they are addressed during the development process.

The AOP is considered as an improvement and extension of the OOP. The OOP can be seen as a successor of structured programming. In OOP, the main entity is the object. An object is a logical combination of data structures and their corresponding methods (functions). Objects are successfully being used as abstractions for passive entities in the real world, and agents are regarded as possible successors of obiects since these can improve the abstraction of active entities. Agents are similar to objects, but they also support structures for representing mental components (for instance, beliefs, goals, actions, and plans). Another important difference between the AOP and the OOP is that objects are controlled from the outside (white box control), as opposed to agents that have autonomous behaviour, which is not directly controllable from the outside (black box control). The OOP has the limitation that it cannot represent complex mental attitudes. Logical programming can represent mental attitudes. It has limitations in the usage of capabilities of action, which can be represented in the OOP. Therefore, it can be presumed that the OOP and the logical programming paradigms merge to form the AOP, in the process of evolution.

IV. JAVA ARCHIEVE (JAR) FILES

The Java Archive (JAR) file make programmer enables to bunch multiple files into a single archive file. The JAR file contains the class files and supplementary resources associated with applets and applications. JAR files are packaged with the ZIP file format, so programmer can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking. These files also support electronic signing feature. To perform basic tasks with JAR files, the Java Archive Tool are used which provided as part of the Java Development Kit (JDK). Java Archive tool is invoked by using the jar command. For creating a JAR file

jar cf jar-file input-file(s)

will be executed. To view the contents of a JAR file

jar tf jar-file

should be executed. For extracting the contents of a JAR file

jar xf jar-file should be executed. To run an application packaged as a JAR file which requires the Main-class manifest header

java -jar app.jar

should be executed.

JAR files support a wide range of functionality, including electronic signing, version control, package sealing, and others.JAR file's manifest gives a JAR file this versatility. The manifest is a special file that can contain information about the files packaged in a JAR file. By tailoring this meta information that the manifest contains, programmer enable the JAR file to serve a variety of purposes.

When you create a JAR file, it automatically receives a default manifest file. There can be only one manifest file in an archive, and it always has the pathname

```
META-INF/MANIFEST.MF
```

When you create a JAR file, the default manifest file simply contains the following:

```
Manifest-Version: 1.0
Created-By: 1.6.0 (Sun Microsystems Inc.)
```

These lines show that a manifest's entries take the form of "header: value" pairs. The name of a header is separated from its value by a colon. The manifest can also contain information about the other files that are packaged in the archive. Exactly what file information should be recorded in the manifest depends on how you intend to use the JAR file. The default manifest makes no assumptions about what information it should record about other files. Digest information is not included in the default manifest. You use the m command-line option to add custom information to the manifest during creation of a JAR file. The Jar tool automatically puts a default manifest with the pathname META-INF/MANIFEST.MF into any JAR file you create. You can enable special JAR file functionality, such as package sealing, by modifying the default manifest. Typically, modifying the default manifest involves adding special-purpose *headers* to the manifest that allow the JAR file to perform a particular desired function. To modify the manifest, you must first prepare a text file containing the information you wish to add to the manifest. You then use the Jar tool's m option to add the information in your file to the manifest. The basic command has this format:

jar cfm *jar-file manifest-addition input-file(s).*

V. BENEFITS OF JAR FILES

The far files play very important role in agent-oriented program code. The JAR file provides many benefits:

- *Security*: By using jar files in agent-oriented programming code, the programmers can digitally sign the contents of a JAR file. Users who recognize programmer signature can then optionally allow programmer software security rights it wouldn't otherwise have.
- *Package Sealing*: Packages stored in JAR files can be optionally sealed so that the package can enforce version consistency. Sealing a package within a JAR file means that all classes defined in that package must be found in the same JAR file.
- *Decreased download time*: If applet is bundled in a JAR file, the applet's class files and associated resources can be downloaded to a browser in a single HTTP transaction without the need for opening a new connection for each file.
- *Compression*: The JAR format allows programmers to compress their files for efficient storage. It provide utilization of memory space.
- *Packaging for extensions*: The extensions framework support the mechanism by which programmer can add functionality to the Java core platform, and the JAR file format defines the packaging for extensions. Java 3D™ and Java Mail are examples of extensions developed by Sun. By using the JAR file format, programmers can turn their software into extensions as well.
- *Package Versioning*: A JAR file can hold data about the files it contains, such as vendor and version information.
- *Portability*: The mechanism for handling JAR files is a standard part of the Java platform's core API.

These benefits are basic applications of using jar files. The programmer can convert the agent program code into jar files.

VI. ENHANCING SECURITY OF PROGRAM CODE

The security of agent program code is very necessary. Without proper security can be modified without concern of the programmer. With help of jar files, the concept of digital signature can be used. The programmers can sign a JAR file with their electronic signature. Users who verify their signature can grant their JAR-bundled software security privileges. The user can verify the signatures of signed JAR files that user want to use. The ability to sign and verify files is an important part of the Java platform's security architecture.

Security is controlled by the security policy that's in force at runtime. The programmers can configure the policy to grant security privileges to applets and to applications. The Java platform enables signing and verification by using special numbers called public and private keys. Public keys and private keys come in pairs, and they play complementary roles. The private key is the electronic "pen" with which you can sign a file. As its name implies, programmer private key is known only to them so that no one else can "forge" their signature. A file signed with their private key can be verified only by the corresponding public key. Public and private keys alone, however, aren't enough to truly verify a signature. Even if you've verified that a signed file contains a matching key pair, users still need some way to confirm that the public key actually comes from the signer that it purports to come from.

One more element, therefore, is required to make signing and verification work. That additional element is the certificate that the signer includes in a signed JAR file. A certificate is a digitally signed statement from a recognized certification authority that indicates who owns a particular public key. A certification authority are entities (typically firms specializing in digital security) that are trusted throughout the industry to sign and issue certificates for keys and their owners. In the case of signed JAR files, the certificate indicates who owns the public key contained in the JAR file. When programmers sign a JAR file their public key is placed inside the archive along with an associated certificate so that it's easily available for use by anyone wanting to verify their signature.

The programmer can use the JAR Signing and Verification Tool to sign JAR files. It can be invoked the JAR Signing and Verification Tool by using the jarsigner command. Before signing a JAR file, the programmer must first have a private key. Private keys and their associated public-key certificates are stored in password-protected databases called keystores. A keystore can hold the keys of many potential signers. Each key in the keystone can be identified by an alias which is typically the name of the signer who owns the key. The basic form of the command for signing a JAR file is jar signer jar-file alias etc. The Jar signer tool will prompt you for the passwords for the keystone and alias. This basic form of the command assumes that the keystone to be used is in a file named .keystone in their home directory. It will create signature and signature block files with names x.SF and x.DSA respectively, where x is the first eight letters of the alias, all converted to upper case. This basic command will overwrite the original JAR file with the signed JAR file.

Typically, verification of signed JAR files will be the responsibility of Java Runtime Environment. The programmer browser will verify signed applets that it downloads. Signed applications invoked with the -jar option of the interpreter will be verified by the runtime environment. However, the user can verify signed JAR files yourself by using the Jarsigner tool. You might want to do this, for example, to test a signed JAR file that you've prepared. The basic command to use for verifying a signed JAR file is:

```
jarsigner -verify jar-file
```

This command will verify the JAR file's signature and ensure that the files in the archive haven't changed since it was signed. The user will see the following message if the verification is successful: jar verified.
If user try to verify an unsigned JAR file, the following message results: jar is unsigned. (signatures missing or not parsable). If the verification fails, an appropriate message is displayed. For example, if the contents of a JAR file have changed since the JAR file was signed, a message similar to the following will result if user try to verify the file:

```
jarsigner: java.lang.SecurityException: invalid SHA1
signature file digest for test/classes/Manifest.class
```

CONCLUSION & FUTURE WORK

The security of agent program code is mandatory. It should be confirmed that the user get the verified code from the programmer. In this paper we have discussed an approach to achieve the security of agent program code by using JAR files. It is very easiest mechanism to create the JAR files. The programmers have no need to use other software & writing more code to implementing the security. There is jar & jarsigner tool available in Java Development Toolkit. One more advantage of using this approach is that the overall process is very simple. This process has fewer steps.

The JAR files enable the programmers to define the single point of entry for execution of the code very efficiently. How much complex the structure of the project, the programmers can define single point of entry by creating manifest file. The security is based on process of signing & verification. The programmer can use own his certificate.

The scope of agent oriented programming is so wide. Hence we can use other more security algorithms to implementing more security.

REFERENCES

[1] C. Guilfoyle and E. Warner “Intelligent agents: the new revolution in software” Ovum 1994.
[2] Y. Shoham “Agent-oriented programming” Artificial Intelligence, 60 (1) 51-92 1993.
[3] N. R. Jennings and M. Wooldridge (eds.) “Agent technology: foundations, applications and markets” Springer Verlag 1998.
[4] S. Russell and P. Norvig “Artificial Intelligence: A Modern Approach” Prentice-Hall 1995.
[5] M. Wooldridge “Agent-based software engineering” IEE Proc. on Software Engineering, 144 (1) 26-37, 1997.
[6] M. Wooldridge and N. R. Jennings “Intelligent agents: theory and practice” The Knowledge Engineering Review 10 (2) 115-152, 1995
[7] P. R. Cohen, A. Cheyer, M. Wang, S. C. Baeg “OAA: An Open Agent Architecture”, AAAI Spring Symposium, 1994.
[8] Wooldridge M “Agent-based software engineering”, IEE Proc. Software Eng. 144(1): 26–37,1997.
[9] Shoham Y “Agent-Oriented Programming, Technical Report” STAN-CS-1335-90, Computer Science Department, Stanford University, Stanford, CA 94305, 1990
[10] Wagner G “The agent-object-relationship meta-model: Towards a unified view of state and behavior”, Inf. Syst. 28(5): 475-504, 2003
[11] Wang L, Guo Q “Mobile Agent Oriented Software Engineering (MAOSE)” In Karmouch A, Korba L, Madeira E (Eds.): MATA LNCS 3284, Springer-Verlag Berlin Heidelberg pp. 168-177, 2004
[12] Lind J. “Issues in Agent-Oriented Software Engineering” in First International Workshop on Agent-Oriented Software Engineering (AOSE-2000), Limerick, Ireland 2000
[13] Conrad, S., Saake, G., and Tuerker, C. 1997. “Towards an agent-oriented framework for specification of information systems”. In Agents'97 Conference Proceedings 1997.
[14] Guerraoui, R. . “Strategic directions in object oriented programming. ACM Computing Surveys” 28, 1996.
[15] B. Hodjat, C. J. Savoie, M. Amamiya, “Adaptive Agent Oriented Software Architecture”, submitted to Pacific Rim International Conference on Artificial Intelligence (PRICAI 98), November 1998.

AUTHORS PROFILE

Mr. Vivek Jaglan obtained his B.E(CSE-2004)from M.D. University, M.Tech(CSE-2008) C.D.L.U Universityand Ph.D (CE) Pursuing From Suresh Gyan Vihar University.. He has attended various national, international seminars, conferences and presented research papers on Artificial Intelligence and Multi-Agent Technology.

Mr. Surjeet Dalal obtained his B.Tech CE (2005) from Kurukshetra University, M.Tech CE(2010) M.D.U University. Currently he is working as Asst. Professor in Department of Computer Science & Engg., Manav Rachna International University Faridabad. He has presented research papers on Artificial Intelligence and Multi-Agent Technology.

Dr S Srinivasan obtained his M.Sc (1971), M.Phil(1973) and P.hd (1979) from Madurai University . He served as Lecturer for 7 years in National Institute of Tehnology in the Computer Applications Department . Later he joined Industry as IT Head for 18 years . Again he started his teaching career serving as Professor and Head of the Department of Computer Science, PDM College of Engineering , Haryana, India. He is member of Computer Society of India.