IOSR Journal Of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, P- ISSN: 2278-8727 Volume 14, Issue 4 (Sep. - Oct. 2013), PP 36-43
www.iosrjournals.org

A Secured Agent-Based Framework for Data Warehouse Management

Nachamada Blamah[1], Aderemi Adewumi[2], Michael Olusanya[2], Asabe Ahmadu[3]
[1]*Dept of Computer Science, University of Jos, Nigeria.*
[2,3]*School of Computer Science, University of KwaZulu-Natal, Durban, South Africa,*
[4]*Dept of Computer Science, Federal Univ. of Tech., Yola,, Adamawa State, Nigeria,*

Abstract: *Managing data warehouses (DWs) is typically characterized by intensive data processing and protracted activities which usually degrade performance. Moreover, DWs are usually designed with the overall objective of making available the content to users, and for them to stay alive, all the management phases need to interact with external and heterogeneous sources. These obviously expose the system to wider security issues. In order to enhance performance, we present a scheme that utilizes the mobility characteristic of agents. The scheme is designed with well-defined communication interfaces within the agent structures in order to improve on the security, where communications within the DW must be done through communicators that require authentications.*

Keywords: *Agent, component, data warehouse, permission, security.*

I. Introduction

A data warehouse (DW) is a collection of data from different sources organized in summarized and aggregated patterns. This is usually composed of the historical data of an organization stored for end user analysis. The data stored in the warehouse are uploaded from the operational systems and may pass through an operational data store for additional operations before they are used in the DW for reporting. The detailed decision making warehouse is normally maintained separate from the organization's operational databases because the functional and performance requirements of the DW are quite different from those of the operational databases.

Data warehousing technologies have been successfully deployed in many industries including retail, manufacturing, telecommunications, financial services, healthcare services and so on. They all have a common set of characteristics: sheer size, interrelated data from many (often) distributed and heterogeneous sources, and access by hundreds and thousands of people [5].

Because of the importance that is attached to DWs, different approaches [3, 8, 9, 10] for managing them have been proposed, with their pros and cons. We therefore present the background and motivation to this research.

Literatures have looked at data warehouse (DW) and security from different perspectives. The introduction of a virtualized proxy component was proposed as a method of improving the security of distributed DW systems [4]. This component hides the physical address of the DW and acts as a middleman between the clients and the DW server. The proxy serves as the place holder for a set of security enforcement. [19, 21] presented multiagent systems models that operate in distributed systems and warehousing environments, with emphasis on the multiagency, while attention was not paid to security issues. The approach described in [23] explains a method of representing security and audit rules for DW at the logical level by using common warehouse meta-model. This is achieved by the extension of the relational model to consider security and audit measures represented in the conceptual modeling.

As pointed out in a survey on currently available data security techniques and security requirements [12, 22, 24], not much has been accomplished in totally securing DW systems and at the same time giving an excellent performance. [6] presented an approach to balance security and performance to enhance the privacy of DW. In this approach, the emphasis is on protecting sensitive business data in DWs that balances security strength with database performance. [1, 16] offered a model driven approach to the development of secure DWs based on XML and UML, while [17] presented a hybrid approach to an improved security framework for DW, which considered the implementation of filter over data before they are kept in statistical form in a DW.

Agents systems security have been considered [2] in literature as well. [11] proposed a protocol which provides an integrated solution for the various types of attacks. The protocol used trip markers, digital signatures and encryption techniques to make the agent immune to most types of security threats. We have researches that require agent platforms to maintain execution traces of an agent's code [14], agents to maintain a record of the agent platforms visited [18], convey proof of safety properties of code [13], secure recording of itineraries through cooperating agents [15], and single hop solution with agents limited to having a single hop from their home platform [7].

But the DW techniques described will only be successful under stable network condition. In our approach, a component-based security to agent based data warehouse is presented.

II. Background And Motivation

Organizations typically maintain the detailed decision making warehouse separately from their operational databases. This is done because the DW supports on-line analytical processing (OLAP), the functional and performance requirements of which are quite different from those of the on-line transaction processing (OLTP) applications traditionally supported by the operational databases.

The characteristics that distinguish OLAP from OLTP applications and the typical client-server architecture based on Remote Procedure Call (RPC) or message passing *over networks* generally pose great challenges to the management of data warehouses. From the Extract-Transform-Load (ETL) process to the view maintenance and the data access phases, DW activities are quite data intensive.

In a highly loaded network with lots of interactions and requirements for high throughput, the waiting/blocking in both message passing and RPC over the network may be unacceptable for some applications due to delays. Throughout the waiting period, resources over the network that could have been used by others are tied down. Although there are other techniques that try to avoid the blocking of processes, nonetheless communicating processes still require network connections and resources. These techniques generally lack the ideal autonomy that is required by very large systems like a DW, as such, a mobile agent based approach to DW management is suitable as a remedy to these challenges, where agents are used to coherently manage the entire system in order to give continual and uninterrupted access of the appropriate content to users.

Agent technology as solution to distributed problem solving is faced with the challenges of attacks among the various components. DW is also usually an open system that is accessible by many users, with the overall objective of making available the content of the warehouse; the ETL, management, and data access phases need external and heterogeneous sources of data to keep the warehouse alive. These obviously expose the system to wider security issues. In this paper, we address the security challenges within agent-based DW.

III. Overview Of Our Agent System For Data Warehouse Management

A mobile agent is not bound to the system where it begins execution; it has the unique ability to move from one system in a network to another. The ability to travel lets a mobile agent move to a system that contains an object with which the agent wants to interact and take advantage of being in the same host or network as the object, thereby reducing the need for communicating over the network. When an agent travels, it transports its state and the code with it. Mobile agent technology allows processes to migrate among different machines.
To enable the efficient management of DW based on mobile agents, a high level architecture that covers the activities across the entire warehouse management process has been designed, which specifies the functions that different agents perform within warehouse environment. The components that make up the architecture are categorized into four major segments: the operational systems, data factory, warehouse, and data access. This is shown in Fig. 1. The dashed arrows in the figure indicate the presence of network, which connects only when there is a need for the components involved to communicate. On three of the components, we have the database connectivity interface that enables the mobile agents to get access to the data sources.

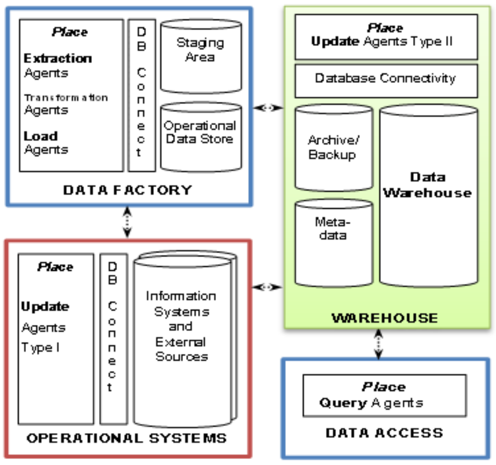

Figure 1: Architecture for secured agent-based data warehouse management

***Operational Systems Component*:** This component houses the OLTP systems where the detailed day-to-day transactions take place. On this component, we have the databases that store operational data of an enterprise.

We then have mobile agents called Update Agent type I (UAI), residing on this component. These agents monitor updates on the data sources that affect the materialized views on the DW and then report such updates on the DW. They are useful in both the *self-maintainable* and the *not self-maintainable* categories of DW maintenance. UAI also supplies data to the operational data store (ODS), an item on the Data Factory component that is responsible for keeping prepared data for the DW in advance.

Data Factory Component*:* This component contains the entities that form the core technology for the mobile agent-based ETL process. This is the data refinery, a place for the initial preparation of data before they are finally taken to the DW. Three types of agents are initially deployed on the Data Factory component, they are: Extraction Agent (EA), Transformation Agent (TA), and Load Agent (LA). EA and LA are mobile agents, while TA is stationary agent. The Data Factory initially dispatches the EAs to the operational systems for the purpose of extracting unclean data. On getting to the operational systems, they connect to the data sources through the use of JDBC interface and perform the necessary extractions. The TAs are responsible for the transformation of extracted data, and the LAs do the final loading of the transformed data.
Warehouse Component: The Warehouse component houses the DW, where we have data in detailed and lightly/highly summarized form. This is the repository and the final target for all the data collected and prepared from different sources.

Update Agent type II (UAII) initially resides on the Warehouse component. This agent is in charge of updates in the *not self-maintainable* category of DW view maintenance. When an update occurs at a remote data source, the data source only notifies the DW of the updates through UAI. The DW then issues UAII with requests to the data source(s), requesting all the relations required in the view definition. After receiving UAII with the requests, network is disconnected and requests are processed offline, or other network nodes utilize the network resources. The remote data source(s) then process(es) the request, after which the results are sent to the DW through UAII after reestablishing network connection.

Since the DW would be the main target for any malicious code, we add more security by restricting access to this component such that only certain types of agents get admittance – which must supply necessary credentials before gaining access to the DW. The agents that are permitted to access this component are the LA, UAI, UAII and the Query Agents (QA) from the Data Access component.

Data Access Component: The Data Access component is concerned with getting access to the data on the DW. It provides facilities for reporting query, application development, OLAP and data mining tools. Requests are made to the DW through the means of the Query Agent (QA) that resides on the Data Access component. When a request is issued, the requesting client goes offline while the results of the requests are being processed at the DW. When the results are ready and network connection is reestablished, the results are presented to the requesting client.
When any of the types of agents described above is overloaded with a task, we permit them to spawn a light weight agent (LWA) of their types to render assistance.

IV. The Framework

Since the mobile agents are designed to execute within a DW setting, one key non-functional requirement of the mobile agents system is to design a reliable framework such that the DW can operate in the face of intermittent network connections. In addition to that, security requirements demand appropriate measures to be in place to protect the system from attacks.

In this section, we present the definitions of various entities that make up the security framework, which enables agents to successfully execute within a DW setting.
We define the entities within the DW with which agents interact as a tuple $\xi = (V, r)$, where V is a materialized view within the system and r is a source relation.
The warehouse can be in any of the following state: $\omega = (safe, suspicious, unsafe)$. A safe state is when the warehouse is perceived not to be in an unsecured state. The warehouse is in a suspicious state when there is a doubt about the execution of agents within the servers or other components communicating with the system, and hence, the executing components may be terminated. An unsafe state is when the warehouse has been certified to be in a vulnerable state.
The actions that agents can perform are the normal database operations, and is defined by $\alpha = (append, delete, update, browse, \ldots)$.
These actions are produced by agents which execute within the DW. An agent is therefore defined as $A : R^{\omega} \rightarrow \alpha$, where R is a run that leaves the warehouse in a state ω; that is, it is as a result of the agent's action that the execution takes place within an environment
We now formally define the set of DW agents as $Ag = \{U_a^I, U_a^{II}, E_a, T_a, L_a, Q_a\}$,
where

U_a^I is the update agent type I
U_a^{II} is the update agent type II
E_a is the extraction agent
T_a is the transformation agent
L_a is the load agent, and
Q_a is the query agent.
We define a place as an executing platform for agents, which is represented by the power set of agents as $P = 2^{\{A_i\}}, i = 1, \ldots, n$, where n is the total number of agents within the place. This means that a *place* can contain no agent $\{\emptyset\}$, or a set of agents in several combinations. A blackboard is attached to each place so that moving agents can leave traces of their subsequent destinations.
To make the communicating entities more secured, a component-based communication was adopted where the agent system's communication infrastructures are inbuilt within the servers and the agents. There are three types of objects that can engage in communication: the Servers, the Agents, and the Users. Communications can take place locally or remotely, depending on the communicating objects.
To enforce more security, we identified and permit only four possible communication patterns among the components within the system, so that all other communication patterns are mapped into these four. They are:

- Agent/Server communication: Communications between the server and agents must be done locally; that is, an agent must be executing on the server it wishes to communicate with. Typically, the type of interaction that takes place here is a local procedure call, where the two communicating objects reside on the same machine.
- User/Server communication: Users get access to the server through a local user-server interface. The users specify what they need through the interface and submit the request. If the request is to interact with a server, then that server must be on the same machine where the request is submitted.
- Server/Server communication: The only communication that can take place remotely is a server to server communication. When a server on one machine wants to talk to another server on another machine, the requesting server connects the network and makes the request remotely; for example, when a mobile agent requests the current server to transport it to another server, the requesting server connects to the destination server and issues a request remotely. In our implementation, the *rmi* protocol was used.
- Agent/Agent communication: Mobile or static agents communicate with other mobile or static agents only locally. This means that if two agents that are not on the same machine want to communicate, one of them must move (or spawn and send a light weight agent (LWA) messenger of its type) to the machine where the other agent is, and then communication would take place locally. The mailbox approach as an agent-agent communication mechanism is explained shortly.

The remaining patterns of communication among the components are made possible as follows:

- User/Agent communication: When a user wants to communicate with an agent, s/he does so through the server. The agent must be on the local machine where the user makes the request, and communication with the agent takes place through the user-server interface. When the user passes a message to the server, the server communicates with the agent as in a agent-server communication. But if the agent has moved to another machine, a request is initiated by the server by creating a LWA and sending it with the message to locate the agent in question, and communication takes place in the same way an agent would communicate to another agent locally (on the same machine).
- User/User communication: a user makes use of the server to communicate with another user on the same or different server.

The agent-agent and server-server communications require more specialized mechanisms; user-server communication is simple, since they are performed locally through dialog prompts, and agent-server communications are simply done by local method invocations.
These patterns give rise to the definition of four permissible communicators on the components within the system. We formally define a $communicator$ as a 4-tupple:
$C = (A_s, U_s, S_s, A_a)$
Where:
A_s is the Agent/Server communicator component,
U_s is the User/Server communicator component,
S_s is the Server/Server communicator component, and
A_a is the Agent/Agent communicator component.
We now define a $server$ as the component that allows the execution of agents by providing the necessary services. Therefore, we present server as a tuple $S = (C^*, P^*, \xi^*, B)$, which contain

$communicators, places, entities$ and a $blackboard$. The asterisk (*) indicate a vector containing at least one item.
We define a data warehouse to be a set of servers, $W = \{OS_S, DF_S, WH_S, DA_S\}$, which are the operational system server, data factory server, warehouse server, and the data access server respectively. These items represent the components earlier described in Fig. 1, which exist on the entire warehouse system.
For any agent to execute within the system, permission is required to ensure that the warehouse remains in a safe state.
We therefore define a $permit$ as a 5-tuple:
$\rho = (A, \alpha, \xi^*, S^*, Ag)$ where the agent A with actions α is to access ξ^*entities and S^* servers in order to interact with Ag agents. An agent A is granted a permit ρ if only its actions α will not take the warehouse into an unsafe state.

V. Implementing The Framework

The proposed framework was implemented in java and MySQL database, and we present the modules as *Agent*, *Communication*, *Places,* and *Server* subsystem.

1.1 Agent Design

An agent's class structure was designed that represents all the agents on the architecture of Fig. 1. All the classes, with the exception of the Authenticate class, are located in the *agent* package – the Authenticate class is found in the *security* package.

Within the architecture, we have static and mobile agents. Both types of agents are declared as abstract classes that implement the *Agent* interface. These are related to communicators to enable communication.
The mobile agents, before migrating to any machine, requests the server where it is currently executing to transport it to the desired machine. The server then makes the requests on behalf of the mobile agent, which performs the transfer to the desired destination, if the request is granted. This class relates the *Authenticate* class, which is used for the purpose of authentications by agents.
Mobile agents are further categorized into single hop mobile agents and multi-hop mobile agents. The single hop agents are those whose functionalities do not require them to travel more than one machine, as such this is incorporated into the design for the purpose of security.
The Load Agent (LA) is the only single hop agent in the system. This is declared as a *final* class in order to prohibit extension; this is done for security purpose, which is explained in more details under agent security system.
Multi-hop mobile agents can hop among several machines and have the capability of deciding which machines to visit next. All other agents in the system are multi hop agents, and they are declared as *final* classes in order to prohibit extension.

1.2 Communication components design

The mailbox approach described in [20] was used for the implementation, and what follows is the entire system's communication structure, with the class hierarchy displayed in Fig. 2.

Server-server communications can be performed both locally and remotely. Two interfaces are therefore defined which qualify derived classes to communicate with the server either locally or remotely. These interfaces are called *LocalCommunicator* and *RemoteServerCommunicator* to represent local communications and remote server communications respectively. The *LocalCommunicator* defines all local communications for other types of objects within the system; therefore agent-agent, agent-server and user-server communicators use this interface as well. These classes are in the *communicator* package.
As depicted in the Fig. 2, four abstract classes are defined to implement the LocalCommunicaor interface; *AgentAgentCommunicator*, *UserServerCommunicator*, *AgentServerCommunicator*, and *ServerServerCommunicator*. Each of these classes is related to the *Authenticate* class so that each time communication is about to take place, authentication must be performed first. These classes provide the general functionalities required for local agent-agent, user-server, agent-server and server-server communications respectively. At the bottom of these abstract classes, we define the DW specialized classes that extend each of the classes.

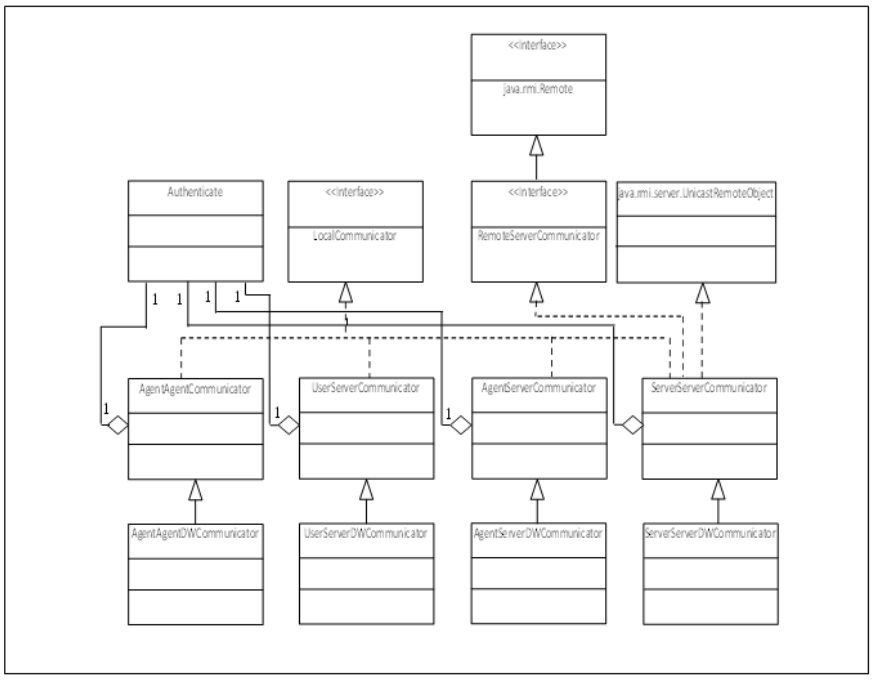

Figure 2: Communicators' design (UML Class Diagram)

The server communicates remotely through the *RemoteServerCommnicator* interface (which provides access to all the remote methods), as such this interface must extend the *java.rmi.Remote* interface. It also extends the *java.rmi.server.UnicastRemoteObject* class for marshaling and unmarshaling remote references to objects.

1.3 Place Design

Places are execution environments where agents execute on servers. A *Place* in general provides the agents that execute within it with the services of the host machine. The *Place* class hierarchy is located in the *server* package.

Mainly there are seven classes that extend the *Place* abstract class. These are the *MonitorPlace*, *TransactionPlace*, *SubmissionPlace*, *WaitingPlace*, *MaintenancePlace*, *EnquiryPlace*, and *ReturnPlace*. These correspond to the places on the high level architecture described in Fig. 1.

1.4 Server Design

Agents execute within places on servers. The servers instantiate place(s) in order to execute agents, and they must also instantiate communicator(s) to enable them communicate with other entities. A *Server* provides blackboard where agents record information about their next destinations so that they can be traced. The server classes are stored in the *server* package.

The main server class is called *AgentServer*, which defines all the functionalities required by all kinds of servers, and four kinds of servers exist within the system which extend the main server class. Each of them resides on the different components of the DW architecture and are referred to as *OperationalSystemServer*, *DataFactoryServer*, *WarehouseServer*, and *DataAccessServer*. Each of these classes relates to at least one *Place* class. The *Place* class is an execution environment for all agents within the system. A *Place* generally provides the services of the host machine to the agents that execute within it.

VI. Experimentation And Discussion

An experiment was conducted to integrate the different components and see the behavior of the network during the extraction, transformation and loading (ETL), updates of self- and not self-maintainable views, and query processes. A demonstration of what is obtainable from the system follows, with the following computer setting: a Pentium M with 1.8GHz, 80GB/512MB running WinXP SP2, java (J2SE) and MySQL server5. Then a Pentium IV with 3.2GHz, 150GB/1GB also running WinXP SP2, java (J2SE) and MySQL server5.

Table 1 explains the calculations. In all the cases, an average size of 260byte-records were considered. A case of 150 records was considered for the extraction, transformation and loading phase, while for the two types of maintenance, a case of 100 records each was considered, and lastly for the user query, a case of 50 records was considered. The time for executing each process is as indicated, and the average time per record and average time per byte are also presented.

TABLE I. COMPARISONS OF METHODS

Activity	**Agent based system**			
	Time (s)	*No. of Records*	*Ave. time per record*	*Ave. time per byte*
Extraction	3.2	150	0.021333	0.082 x 10-3
Transformation	92	150	0.613333	2.359 x 10-3
Loading	6.8	150	0.045333	0.174 x 10-3
Self-maint.	3.4	100	0.034000	0.131 x 10-3
Not self-maint.	5.6	100	0.056000	0.215 x 10-3
Query	0.55	50	0.011000	0.042 x 10-3
	Client-server based system			
Extraction	5.4	150	0.03600	0.138 x 10-3
Transformation	155	150	1.03333	3.974 x 10-3
Loading	7.7	150	0.05133	0.197 x 10-3
Self-maint.	4.7	100	0.04700	0.181 x 10-3
Not self-maint.	8.5	100	0.08500	0.327 x 10-3
Query	0.69	50	0.01380	0.053 x 10-3

The achievement in time savings is obviously as a result of transactions that were mostly performed offline in the agent based approach, while the client-server made use of network infrastructure for the transactions.
The first security concern we are addressing is the attack of agent against platform. All platforms certify agents to be of the required type (EA, QA, UAI, UAII, TA) before allowing them to execute; i.e., any foreign agent that does not fall within these agent types need to be certified by the administrator before it is added to the trusted parties' list. The *instanceof* keyword is used as follows, to certify that object x is of class type y: (x *instanceof* y), which must be an agent within the set $\mathrm{Ag} = \{\mathrm{U}_a^{\mathrm{I}}, \mathrm{U}_a^{\mathrm{II}}, \mathrm{E}_a, \mathrm{T}_a, \mathrm{L}_a, \mathrm{Q}_a\}$. This is carried out within a class called Authenticate, which authenticates agents and servers.
Next we prohibit the extension of the base classes of the Agent's class. We make the following analysis:
1: If the extensions of these classes are allowed to be valid objects to execute, an attacker can extend any of these classes and then define some malicious codes within the new class.
2: If a class or an interface type at a higher level is allowed to be a valid object to execute, an attacker can also extend this class or implement this interface and then define some malicious codes within the new class.
Therefore we ensure that x is always a class at the lowest level of the agents' class hierarchy of the agents' design; not the extension of any of the lowest classes in the hierarchy and not any class or interface above them in the hierarchy; this is achieved by using the *final* keyword to prohibit extending the classes.
But enforcing such restrictions would prohibit attackers from extending any abstract class or implementing an interface at a higher level (with the intention of making this a valid class to execute). By doing this, there is no fear of any attack on the platforms from visiting mobile agents.
Finally we need to handle the attack of other platforms and mobile agents against agents. Since all DW agents start executing from certified and trusted platforms, there is no fear of any malicious act from any platform or agent at the starting point.

The design of our communication system is fashioned in such a manner that all communications must go through certain interfaces. These interfaces, called communicators, are protected and placed on trusted servers such that only authorized parties that provide the necessary credentials get access to the communication system. The security of the communicators, and hence of the entire mobile agent system, are completely handled by the security mechanisms of the trusted servers. If an agent wants to move to another platform, it requests the present server (which must possess authorization rights) to transfer it. The current server then establishes a connection and performs the necessary authentication of the second server before handing over the agent, if it deems it fit. While this agent is executing on the foreign platform, the agent communicator will always require authentication from this platform and/or other executing agents (using the *Authenticate* class) before it will give access to its functionalities. In this way, the mobile agent avoids the effects of attacks from both the platform and other agents.
Since it is only the servers that can perform remote communications, and because they always perform authentications before allowing access to their services, attacks from other entities against the mobile agent system is taken care of.

VII. Conclusion

A secured agent based approach to data warehouse management has been presented where agents are used to handle all the management activities of data warehouse right from the extraction, transformation and loading, to maintenance and then to data access phases. In order to enhance performance, the mobility characteristic of agents was utilized to design more robust warehouse architecture within a distributed environment. A scheme with well-defined communication interfaces within the agent structures and with inherent security is designed to handle the security flaws, where communications within the data warehouse must be done through communicators that require authentications.

References

[1] V. Rodolfo, E. Fernández-Medina, M. Piattini, and J. Trujillo, "A UML 2.0/OCL Extension for Designing Secure Data Warehouses," Journal of Research and Practice in Information Technology, 38(1), pp.31-43, 2006.

[2] N. V. Blamah, G. Wajiga, and A. S. Ahmadu, "The Security of Mobile Agents," In Proceedings of the International Conference on NTMCS, Covenant University, Ota, Nigeria, 379-392, 2006.

[3] S. Chen, "Efficient Incremental View Maintenance for Data Warehousing," PhD Dissertation Submitted to the Faculty of The Worcester Polytechnic Institute, UK, 2005.

[4] Y. Qu, and W. Yang, "Improving the Security of the Distributed Enterprise Data Warehouse System," Intelligence and Security Informatics, IEEE International Conference, p.227, 2011.

[5] O. Idika, "Data Mining and Data Warehousing," A Publication of the Digital Bridge Institute, International Center for Advanced Communication Studies, Abuja, Nigeria, 2005.

[6] R. J. Santos, J. Bernardino, and M. Vieira, "Balancing Security and Performance for Enhancing Data Privacy in Data Warehouses," 2011 International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11, pp.242-249, 2011.

[7] W. Jansen, and T. Karygiannis, "Mobile Agents and Security," NIST Special Publication 800-19, 1999.

[8] A. Javed, and S. S. Rafique, "Data Warehouse Maintenance, Improving Data Warehouse Performance through Efficient Maintenance," Masters Thesis, Lulea University of Technology, Division of Information System Sciences, 2006.

[9] B. Liu, S. Chen, and E. A. Rundensteiner, "A Transactional Approach to Parallel Data Warehouse Maintenance," A Publication of the Worcester Polytechnic Institute, UK, 2002.

[10] S. Lujan-Mora, "Data Warehouse Design Using UML," PhD Thesis, Department of Software and Computing Systems, University of Alicante, 2005.

[11] P. Marikkannu, J. J. A. Jovin, and T. Purusothaman, "An Enhanced Mobile Agent Security Protocol," European Journal of Scientific Research, ISSN 1450-216X Vol.51 No.3, pp.321-331, 2011.

[12] R. J. Santos, J. Bernardino, and M. Vieira, "A Survey on Data Security in Data Warehousing Issues, Challenges and Opportunities," International Conference on Computer as a tool (EUROCON), IEEE, pp.1-4, 2011.

[13] G. Necula, and P. Lee, "Safe Kernel Extensions Without Run-Time Checking," In Proceedings of the 2nd Symposium on Operating System Design and Implementation (OSDI '96), Seattle, Washington, USA, 229-243. 1996. <URL: http://www.cs.cmu.edu/~necula/papers.html>

[14] J. J. Ordille, "When Agents Roam, Who Can You Trust?," Proceedings of the First Conference on Emerging Technologies and Applications in Communications, Portland, OR, USA, 1996. <URL: http://cm.bell-labs.com/cm/cs/doc/96/5-09.ps.gz>

[15] V. Roth, "Secure Recording of Itineraries through Cooperating Agents," Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, INRIA, France, 147-154, 1998. <URL: http://www.igd.fhg.de/www/igd-a8/pub/#Mobile Agents>

[16] B. Vela, C. Blanco, E. Fernandez, and E. Marco, "Model Driven Development of Secure XML Data Warehouses: A Case Study," EDBT '10 Proceedings of the 2010 EDBT/ICDT Workshops, ACM publication, Article No. 10, 2010.

[17] S. Ahmad, and R. Ahmad, "An Improved Security Framework for Data Warehouse: A Hybrid Approach." In IEEE Publication, 978-1-4244-6716-7/10, pp.1586-1590, 2010.

[18] G. Vigna, "Protecting Mobile Agents through Tracing," Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, Jyväskylä, Finland, 1997. <URL: http://www.cs.ucsb.edu/~vigna/listpub.html>

[19] T. P. Latchoumi, and R. Sunitha, "Multi Agent Systems In Distributed Data warehousing," International Conference on Computer and Communication Technology, IEEE publication,pp.442-447, 2010.

[20] B. Ellis, "OpenVMS Mailboxes: Concepts, Implementation, and Troubleshooting," OpenVMS Technical Journal Vol. 9, 2007.

[21] N. V. Blamah, and A. O. Adewumi, "A multi-agent-based model for distributed systems processing," International Journal of the Physical Sciences, 7(34), pp.5297-5303, 2012. http://www.academicjournals.org/IJPS

[22] N. Stolba, M. Banek, and A. M. Tjoa, "The Security Issue of Federated Data Warehouses in the Area of Evidence-Based Medicine," Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06), IEEE Computer Society, pp.329-339, 2006.

[23] E. Soler, R. Villarroel, J. Trujillo, E. Fernandez-Medina, and M. Piattini, "Representing security and audit rules for data warehouses at the logical level by using the Common Warehouse Metamodel," Proceedings of the First International Conference on Availability, Reliability and Security (ARES'O6), IEEE Computer Society, pp.914-921, 2006.

[24] E. Soler, V. Stefanov, and J. Maz´on, "Towards Comprehensive Requirement Analysis for Data Warehouses: Considering Security Requirements," The Third International Conference on Availability, Reliability and Security, IEEE Computer Society, pp.104-111, 2008.