MODELING THE AUDIT IN IT DISTRIBUTED APPLICATIONS

Victor-Valeriu PATRICIU
PhD, University Professor
Department of Computer Engineering
Military Technical Academy, Bucharest, Romania

E-mail: vip@mta.ro

Calin Marin VADUVA
Technical University of Cluj-Napoca, Cluj-Napoca, Romania
Co-author of books: Security in Electronic Commerce (2000); Java Programming (1999), Theory of Information Transmission - Laboratory Course (1996)

E-mail: calin.vaduva@fortech.ro

Octavian MORARIU
Student, Technical University of Cluj-Napoca, Cluj-Napoca, Romania

E-mail: morariu.octavian@gmail.com

Marius VANCA
Bachelor Degree form Babes-Bolyai University, Cluj Napoca, Romania
Fortech, Cluj Napoca, Romania

E-mail: marius@fortech.ro

Olivian Daniel TOFAN
PhD Candidate, Assistant Researcher, Department of Mathematics and Computer Science
Babes Bolyai University, Cluj Napoca, Romania

E-mail: to27959@yahoo.com

Abstract: *Quality in software is always an important and forever an "in vogue" topic, especially if we talk about complex distributed IT systems. In the context of the software quality, reliability of the software is a fundamental aspect. If we are talking about critical software solutions, we may not imagine any failure in the software, we may not imagine that some data has been lost or some operation has been done in the wrong way. In this context an important feature that should be build in the software is the audit capabilities of the software. Any application that adheres to a certain quality level must implement a solid audit module in order to be compliant with modern standards. There is no way to prove that the software operates in the right way except auditing the important actions executed by the software. The aim of this paper is to define the requirements for auditing and to propose a solution for implementing them in a software system. The paper starts from the description of the requirements for audit, goes on with a presentation of original concepts in the field and presents in the end the practical approach for implementation of the solution in a real software system.*

Key words: *modeling; IT audit; distributed applications*

1. Introduction

The audit module of a Software System provides the means to record all actions performed both by the direct users of the system and by the system itself. A complete audit system must record not only the actions that were performed, but also the states of the objects affected by those actions. All the information is stored in a chronological way, so tracking and rollback are always possible on system object states. The audit module is a vital part of reliability and security the system as it provides the information regarding all the actions and modifications performed on objects in a structured manner, unlike logging.

From the audit point of view, an **action** is defined as *a specific piece of functionality* of the software system. Execution of an action implies some kind of processing or a transformation to be applied on a dataset. An **object** will be defined as a *set of attributes*, where an attribute is a unique name-value pair.

2. Audit requirements

There are two types of requirements for an audit module. The first type is represented by the functionality requirements. Any audit module should implement the following minimum requirements:

- Record actions performed trough the system
- Record object states
- Provide means to retrieve audit data
- Provide means to track the history of an object
- Provide means to detect any external change of data

Along with these functional requirements there are some security related requirements:

- Audit data must be stored in a secure manner
- Continuous audit must be assured, no gaps allowed
- Compliancy and integration with standards

In the following section each requirement will be discussed for a better understanding.

Record actions performed trough the system represents the base feature for an audit module. This requirement is implemented even in the simplest logging systems. However, if with the logging systems (used mostly for debugging) there isn’t a universal format, for an audit system the information recorded should be consistent. This requirement raises a challenge for application architects as it usually require a common point for all actions performed by the users, or by the system itself. This common point can be defined as a dispatcher. A well-designed application usually contains such a dispatcher, used also as main authorization point.

In order to be able to have a history tracking feature and a rollback option for a software system, the audit module must **record all object states**. When an object is modified by an action there are usually three states to be recorded: initial state (OBJ_PRE),

Vol. 2 No. 1 Spring 2007

ideal state (OBJ_IDEAL) and result state (OBJ_RES). The initial state represents the state of the object before the operation is performed. The ideal state is the final state of the object if the operation is successful. The result state is the final state of the object, regardless of the result of the operation. This document is assuming that the operations, like in real world, can have tree outcomes: successful, failed or partially successful.

At all times the audit module must **provide means to retrieve audit data.** A user of the software system, having the necessary authorization must be able at any time to query the audit module for information regarding actions and objects.

The audit module must also **provide means to track the history of an object.** This implies searching all the saved states and actions related to a specific object, and build a chronological report based on the obtained data-set.

The audit should **provide means to detect external changes of data**. The data managed by the software systems is usually stored in a data base or in other information repository. This means that if someone has access direct to this repository may change directly the information. It is practical impossible to audit these operations as long as they may not be intercepted by our audit system, but the audit module should detect if such a "break" appears.

The most important security requirement for an audit module is to **store the audit data in a secure manner.** The audit module must guarantee that only authorized users can access the information stored. Also, no external modification on the audit data must be permitted. In this way the audit data will be authentic.

Continuous audit must be assured. During the whole life cycle of the application objects a full audit must be maintained. The audit module can allow no gaps in the chronological line.

3. Architecture

There are two solutions for implementing an audit module:

- First one uses hooks to integrate in an given application, intercepting the strategic calls, also known as an external audit system;
- The second one provides a set of interfaces and relies on the application to explicit invoke audit methods, also known as an internal audit system.

Even if this article focuses on the last approach, we may consider this model as a base for implementation of the first approach. One aspect that has been considered by us in the future is migration from the second approach to the first approach using the ideas/concepts from aspect programming. Nevertheless this would be the subject addressed by further work.

In this section is presented the high-level architecture on an audit system. Each sub-module will be presented in detail later in this article. The diagram will show the main interfaces of the audit module and also the relations between sub-modules.

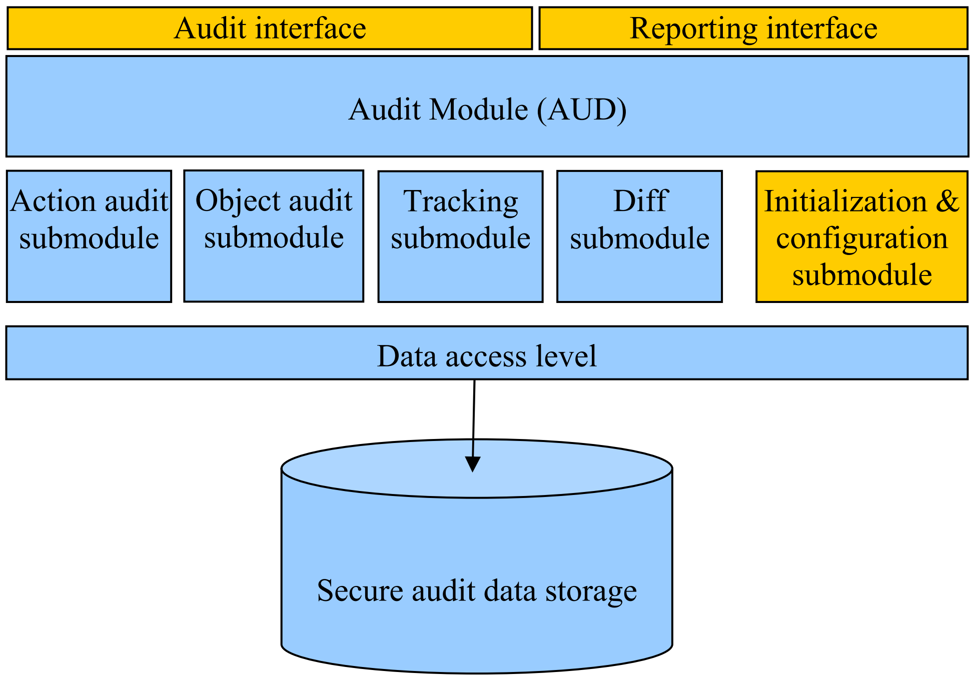

Figure 1. Audit Module Architecture Diagram

The audit module will record two types of information:

- **Action execution**: the user U has executed the action A at time T. Along with this basic information, that should identify an *action type* audit entry, some other details need to be recorded: context in witch the action was executed (network information, application information), a possible reason or description for the action and the outcome of the action (successful, failed or partially successful).
- **Object modification**: if an object is modified at some point, regardless of the initiator of the modification (a user, a batch job, another application), three states of the object must be recorded as defined in the previous section. There are cases when not all three states are relevant, for example if an object is created or deleted. In those cases the OBJ_PRE and the OBJ_RES states will be empty.

For a better understanding of the audit sub-modules, the following section will describe in more detail the functionality for each:

- **Audit interface** –This interface is providing the methods for recording an action execution or an object change. The application's main dispatcher will basically use this interface. As presented at the beginning of this section, this article is assuming that the audit module will be developed as a part of the software system. Considering this approach, an explicit call to the audit module will be required for recording and action execution or the modification of an object. Considering that all kinds of actions and objects in the application must be recorded, this interface must be flexible enough to cope with this requirement.
- **Reporting interface** – This interface will expose the reporting functionality provided by the audit module. In production systems the quantity of data stored and managed by the audit module will be huge. Usually the main problem in audit reporting is filtering the relevant data from the whole amount of recordings. The audit module

must be able to generate reports based on all kind or criteria: all actions for a specific user, history of a specific object, and so on. This interface will also provide the means to retrieve the differences in object states at some given times.

- **Action audit sub-module** – implements the audit interface, providing the methods for recording action related events. Information about the user that executes the action and the outcome of the action will be added to the recording at this level.
- **Object audit sub-module** – implements the persistence related business for the objects before and after a modification is performed on the object. Two methods will be implemented at this level: *auditBefore(operation,OBJ_PRE,OBJ_IDEAL)* and *auditAfter(operation,OBJ_IDEAL,OBJ_REAL)*. As another responsibility for this sub-module we can mention the possibility of data encryption/decryption for stored objects. We must consider a sever performance penalty because of this requirement. This must be considered from the design stage because may affect considerably the reporting feature. .
- **Tracking sub-module** – this module will implement the standard queries to retrieve the most relevant information. Along with this standard reports this module must be able to perform user defined, custom, queries and reports.
- **Diff sub-module** – for a specific object this module will be able to identify and report the differences between two states of the object at given moments in time. This module will be responsible for retrieving and comparing the object states.
- **Initialization and configuration sub-module** – An important part of an audit system is the configuration and initialization section. A tight integration is required with the host operating system for performance and standard compliance reasons and so a strong initialization and configuration module is required. This module will manage the credentials for secure connections to the storage systems and will implement all the high-availability features for the audit subsystem. Also, as only some types or objects and only a subset of actions are relevant for being recorded, the configuration module will provide action and object filtering configuration. This module is also responsible to create the database structure use for auditing using the information stored in configuration files (e.g. xml files), information that defines the structure (metadata) of the auditable object.

4. Recording format

This section will define the format required to record actions and objects in the storage system.

Action record format – The following attributes define the format for action type records:

- **actionID** – the unique identifier for the action performed.
- **actionType** – in data management application there are three main operation types: *new* – creation of a new object, *update* – modification of an existing object, *delete* – terminate the life-cycle of an object.
- **timestampStart** – start time for action execution.
- **timestampEnd** – end time for action execution.
- **auditSrcID** – the source module identifier that requested the audit operation.

- **userID** – the unique identifier of the user that performed the action.
- **subjectID** – the unique identifier of the process or task that requested the audit recording.
- **result** – the actual result of the action execution.
- **description** – description of the action.
- **dynamicAttributes** – For greater flexibility a dynamic map of attributes can be specified for some actions. This map will contain name-value pairs with action specific data.

Object record format – Along with actions auditing, related objects would be recorded with a different format. In real-life applications, objects can become quite complex. This record format should be flexible enough to allow saving all kinds of objects. For each object modification three records will be saved:

- **Initial state –** ***OBJ_PRE*** – This is the state of the object just before the operation is executed. The upper level will retrieve the original object state before the operation is executed. This record will be empty only if the operation executed is of type „new”.
- **Ideal state –** ***OBJ_IDEAL*** **–** represents the ideal state. This should be identical to result state if the operation is successful and identical to initial state if the operation fails.
- **Result state –** ***OBJ_REAL*** – represents the result state of the object. This state will be empty only if the operation executed is of type „delete” and is successful.

For implementing this model, each object should have a formal description available. Objects can be implemented as lists of attributes, where each attribute has the following metadata:

- data-type description
- persistence-type description
- value (or multiple values)

Considering these assumptions the format is as following:

- **objectID** – unique identifier for the object.
- **objectType** – type of the object. This will be the link to the data-type description.
- **objectVersion** – version number. Incremented after each operation.
- **globalChangedNumber** – a global change log number.
- **objectData** – multiple fields, name-value pairs.

5. Workflow diagram

The following diagram describes the main workflow for the audit module, for a single operation or for a group of operations.

The audit process can be divided as a sequence of steps, linked together as in the workflow diagram above. From a high level view the audit of one single operation suppose: the audit initiation, execution of the operation and finish the audit. As a continuous audit chronological line must be assured the audit transaction concept will be used during implementation. The transaction facilitates the recording of operations data even in the case of system breakdown and restart.

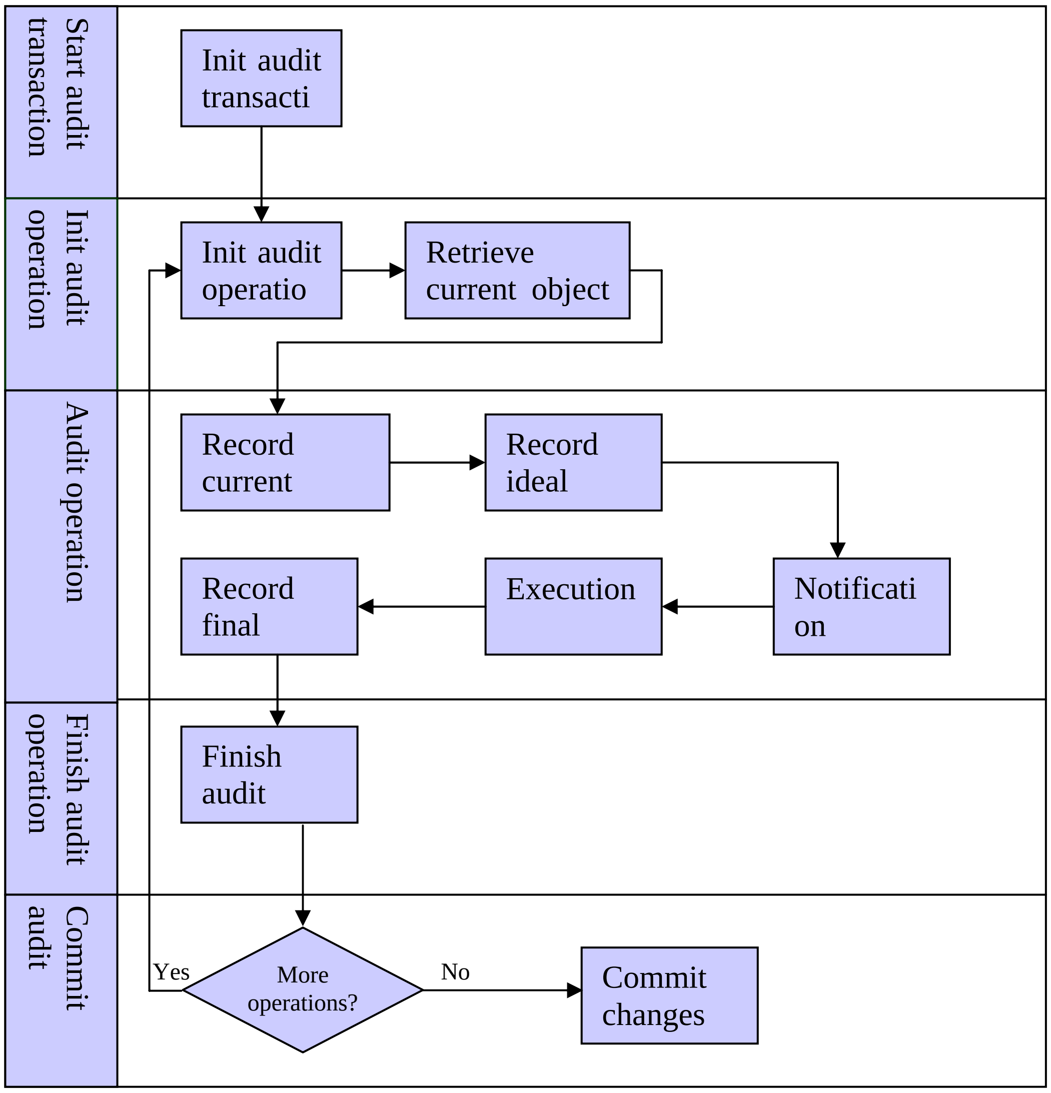

Figure 2. Audit Module Workflow Diagram

As another observation, there are cases when a series of operations are triggered by a single user action (for example the case when the user deletes a list of objects). In this case all operations must be performed in a transaction fashion. The same approach must be extended to the audit functionality, so one audit transaction should be used to record all operations from the group.

The detailed steps from the audit procedure are:

1. Initialize audit transaction. If there is no audit transaction initialized it is assumed that only one operation will be audited. One unique identifier is allocated for this audit transaction to allow later reference to it.
2. For each single operation, the corresponding audit procedure is initialized. One unique identifier is also allocated for this.
3. Once the audit of the operation is initialized, the current state of the operated object is recorded. This suppose: the persistence layer inquiry to obtain the current values of the object properties, create the auditable object and record it in the audit data storage.

4. In case the operation that is going to be performed is a modification of the object, the ideal state is recorded – that is the state of the object which is expected after the operation is executed.
5. At this point the caller should notify the audit module that the execution of the operation will start. The operation start timestamp is recorded.
6. Operation execution.
7. The operation end timestamp is recorded. The obtained object state (for object modification operations) is recorded. The operation result is also recorded.
8. The operation audit is ended.
9. In case a group of operations are audited, the steps 2-8 are repeated for each single operation from the group.
10. The audit transaction is committed.

6. Conclusion

The Audit module of one application keeps a "picture" of the full history of the system life. This recorded information may be used at any point in time to analyze and understand system failures, problems or states, to check the reliability of the system and to assure the application is working and conforming to certain quality standards. It provides a systematic measurable technical assessment of the application. Worldwide companies consider the audit of the systems they use a mission-critical function. The current paper intended to offer a practical approach for implementing such a module covering most of the features requested by a real world application. The solution proposed has not been only presented as a concept but additional has been successfully validated in different distributed applications (n'tier systems, web applications). Nevertheless we may consider that the approach presented may also be improved in the following areas: improve the usability of the module (the developer should integrate the module easier), ensure a high level availability and performance of the audit module. These will be the subject of further research and further practical implementations.

Bibliography

1. **An Implementation Guide for AS/400 Security and Auditing:Including C2,Cryptography, Communications,and PC Connectivity,** Document Number: GG24-4200-00, International Technical Support Organization / Rochester Center, June 1994
2. Chandramouli, R., Sandhu, R. **Role-based access control features in commercial database management systems,** Proceedings of the NIST–NSA National (USA) Computer Security Conference, 1998, Pages 503–511.
3. Ferraiolo, D., Kuhn, R., Chandramouli, R. **Role-based access control,** Artech House computer security series
4. Gamman, E. **Design Patterns: Elements of Reusable Object-Oriented Software,** Addison-Wesley, 1995
5. Gong, L. **Inside Java 2 Platform Security: Architecture, API Design and Implementation,** Addison Wesley Longman, Inc., 1999
6. **HP OpenVMS Guide to System Security - Security for the System Administrator, Added Protection for System Data and Resources,** Ch. 9 Security Auditing http://h71000.www7.hp.com/doc/732FINAL/aa-q2hlg-te/00/00/80-con.html
7. **Linux Audit-Subsystem Design Documentation**

8. McGraw, G. and Felten, E. **User Authentication and Authorization in the Java Platform Sun JAAS - Securing Java,** John Wiley and Sons, 1999
9. **Oracle9i Database Concepts Release 2 (9.2). Part Number A96524-01,** Ch 24 Auditing http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96524/c25audit.htm
10. Patriciu, V. V. **Securitatea informatică în Unix şi Internet,** Ed. Tehnica, Bucharest, 1998
11. Patriciu, V. V., Pietroseanu, M. E., Bica, I., Vaduva, C., Voicu, N. **Security in Electronic Commerce,** Editura All, 2000
12. Rehman, R. **Intrusion Detection Systems with Snort Advanced IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID,** Pearson Education, Inc. Publishing as Prentice Hall PTR, Upper Saddle River, New Jersey, 2003
13. Tofan, O. D., Vaduva, C. **Permission Management in Distributed Applications,** The 31st International Conference "Modern Technologies in XXI century", Bucharest, November, 2005
14. Vaduva, C., Patriciu, V. V., Tofan, O. D. **Authorization framework for distributes systems,** Proceedings of International Conference, "Communications2006", Bucharest, Romania, 2006
15. Vaduva, C., Tofan, O. D. **Security Architectures in Distributed Systems,** The 31st International Conference "Modern Technologies in XXI century", Bucharest, November, 2005
16. Windley, P. **Digital Identity,** O'Reilly , 2005