Faculdade de Engenharia da Universidade do Porto

Reengineering of a Qimonda Wide Audit Management Application

Carlos Miguel Amorim Sousa

Report of Project
Master in Informatics and Computing Engineering

Supervisor: Prof. Miguel Pimenta Monteiro

2008, July

Reengineering of a Qimonda Wide Audit Management Application

Carlos Miguel Amorim Sousa

Report of Project

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Prof. Ana Cristina Ramada Paiva Pimenta

External Examiner: Prof. Manuel Barbosa

Internal Examiner: Prof. António Miguel Pontes Pimenta Monteiro

31st July, 2008

To my parents who always support me,

To my sister Susana for its encouragement and

To the memory of my grandmother.

Abstract

The majority of the legacy systems, still existing in the companies, were developed with an inflexible architecture which hinders the maintenance and upgrade of those systems. Generally, the difficulties are increased by the usage of obsolete technologies, which are no longer supported and for which specialized human resources are difficult to find.

Qimonda Portugal S.A., follows high quality standards in the production of DRAM memories, which is its area of business. The quality control in the company is made by audits that allow detecting eventual deviations to the required standards besides defining actions to correct them.

The company owns a legacy application that permits the management of audits that has an inflexible architecture and was implemented with obsolete technologies that compromise its maintenance and upgrade. In order to correct these aspects the architecture was redesigned and the application was implemented with the latest technologies available in the market, resulting, thus, in an application flexible, modular, easy to maintain and with a good performance.

This project consisted of reengineering an existing audit management application that had to be analyzed so that the features to be implemented could be identified. Then, several existing application in the company were analyzed to see if they could be included in the application's architecture in order to speed up its development and to support all the functional and non-functional requirements.

The chosen architecture was validated with the implementation of a prototype to which were later added the remain features in order to complete the audit management application.

After the implementation of the application, three analyses related to possible upgrades of the application were done. The first analysis consists of considering the possibility of the implemented application be replaced by the SAP Audit Management application. Secondly, the viability of adding new features in the application was analyzed. Those features would allow managing action plans. Finally, the substitution of the existing mechanism of generation of data reports by an external application, Business Objects, was considered.

To sum up, the replacement of the implemented application by the SAP Audit Management application is not advisable. Nonetheless, the upgrade of the application with the development of features for action plans' management and, eventually, other features related to quality management, is recommended. The tool Business Objects should be used to generate reports with the application's data.

Resumo

Os sistemas legados ainda existentes nas empresas foram desenvolvidos, na sua maioria, com uma arquitectura inflexível que dificulta a manutenção e evolução destes sistemas. Geralmente, estas dificuldades são aumentadas pela utilização de tecnologias obsoletas cujo suporte tende a ser descontinuado e para as quais começam a escassear os recursos humanos especializados.

A Qimonda Portugal S.A., cuja área de actividade é a produção de memórias DRAM, rege a sua produção por elevados padrões de qualidade. O controlo da qualidade na empresa é feito através de auditorias que permitem detectar eventuais discrepâncias relativamente aos padrões exigidos e definir acções para as corrigir.

A empresa dispõe de uma aplicação legada que permite fazer a gestão de auditorias mas esta possui uma arquitectura inflexível e está implementada com tecnologias obsoletas que comprometem a sua manutenção e evolução. Para corrigir estes aspectos, a arquitectura foi redesenhada e a aplicação foi implementada com tecnologias mais recentes no mercado obtendo-se assim uma aplicação flexível, modular, fácil de manter e com bom desempenho.

O projecto consistiu então na reengenharia de uma aplicação de gestão de auditorias existente que teve de ser analisada para se identificar as funcionalidades a implementar. Depois disso, foram analisadas várias aplicações existentes na empresa que poderiam ser incluídas na arquitectura da aplicação de modo a agilizar o seu desenvolvimento e suportar todos os requisitos funcionais e não-funcionais.

A arquitectura definida foi validada com a implementação de um protótipo ao qual foram, posteriormente, adicionadas as restantes funcionalidades de modo a completar o desenvolvimento da aplicação de gestão de auditorias.

Após a implementação foram efectuadas três análises relacionadas com possíveis evoluções da aplicação. A primeira análise consistiu na avaliação da possibilidade de outra ferramenta existente na empresa, SAP Audit Management, substituir a aplicação implementada. Depois disso, foi analisada a viabilidade da introdução de novas funcionalidades na aplicação que permitiriam a gestão de planos de acções. Por último, foi analisada a viabilidade da substituição do mecanismo de geração de relatórios de dados existente por uma aplicação externa, Business Objects.

Como conclusão das análises efectuadas, não é aconselhada a substituição da aplicação desenvolvida pela solução SAP Audit Management. Contudo, é recomendada a evolução da aplicação com o desenvolvimento de funcionalidades para a gestão de planos de acções e, eventualmente, outras funcionalidades relacionadas com a gestão da qualidade. Para a geração de relatórios com os dados da aplicação deve ser usada a ferramenta Business Objects.

Acknowledgements

Many thanks to the MIEIC[1] Director of FEUP[2] during the majority of my course, Professor Raul Vidal, and to the current Director, Professor Augusto Sousa, for their commitment in the organization of the projects.

Many thanks to the Professor Miguel Pimenta Monteiro for the coordination of the project. I also want to tanks Eng. Adélio Fernandes from Qimonda and all the QPT IT PA AS QS[3] team, in particular Eng. Alberto Pereira for its continuous monitoring of the project and Eng. Pedro Couto and Eng. Nuno Felino for their contributions to the project's execution.

[1] Mestrado Integrado em Engeneharia Informática e Computação

[2] Faculdade de Engenharia da Universidade do Porto.

[3] Qimonda Portugal – Information Technologies – Product Automation – Analysis Solutions - Quality Solutions

Index of Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Context

This dissertation describes the development of a project executed in the scope of the Integrated Master in Informatics and Computer Engineering (MIEIC) of the Faculty of Engineering of the University of Porto (FEUP).

The project took place in an external institution named Qimonda Portugal S.A. for a period of twenty weeks under the supervision of Professor Miguel Pimenta Monteiro, professor at FEUP.

Qimonda Portugal S.A., installed in Vila do Conde, is one of the several factories worldwide that belongs to the multinational company Qimonda AG. The business area of the company is the manufacturing of semiconductors with focus in production of integrated circuits for information storage. Thus, memory chips produced by the company are made for several types of devices like computer's memories, mobile phones or graphic cards.

The whole production process is not accomplished in a single Qimonda factory but each factory is responsible for a specific phase of the development of the memory chip.

The Qimonda site in Portugal is designated as the Porto site and besides the production of memory modules it also contains a R&D center where the competence center of information technologies is located. The competence center plays an important role in the IT department which provides all the technological support to other departments. This department is divided in several sections, from which it is important to emphasize the *Product Automation* (PA) section where the student was integrated to execute the project. Although this section belongs to the organization of Porto site, it provides software solutions and support for all sites worldwide.

The project was developed inside the sub-section of Analysis Solutions (AS) and the student was integrated in the team responsible for providing functional support for quality solutions, this is, software solutions used by the people of the quality department.

1.2 Project

The project consists of reengineering the way how audits are tracked in the company. Audits are inspection mechanisms that allow quality assessment within the company and the quality department is responsible for their planning, control, follow-up and, most of the times, for their execution. Thus, in this project, people in the quality department are a fundamental stakeholder because they represent the client and the end users of the software application which will result from this project.

To contextualize correctly the project, it is important to refer that audits are executed in all factories of the company and each one needs a software solution to support their management. Therefore, the end users of the application are not only those that belong to the Porto site but all the people of the quality departments in sites all over the world.

Each site has its own way to monitor audits and, currently, there are several software solutions used for that effect. This situation is being changed with the aim of achieving a more complete integration among all the company sites and sharing the maximum of resources among them including software solutions. If this is applied to the case of the management of audits, it will be possible to obtain a considerable cost reduction and an increase of productivity if the same application is used in several sites. In order to do so, it will be necessary to choose an application among all that exist and to discontinue the usage of the others. This decision has to be made according to several factors inherent to the company's policies and values and, probably, the chosen application will have to suffer some changes in order to be used in all sites.

According to several criteria, an application that was used in Qimonda site in Malacca (Malaysia) was selected to be upgraded in order to be used in other sites, such as the Suzhou site (China) and in the recently created site in Senaï (Malaysia). In the Porto site, the software solution to support the management of audits is still under evaluation. This project consists of doing a complete reengineering to the application. The name of the application, Audit Management, will be maintained after the reengineering process, so it will be hereinafter referred to as Audit Management.

1.3 Motivation and Objectives

The structure of the company evolved in the last years and some changes were done to increase the cooperation and interoperability among the sites. The IT department was also included in this reorganization and some measures were taken to achieve the goals established by the company. One of these measures was the sharing of information and resources which lead to the conclusion that the company owns several different software applications to perform the same functions and with the same features. In certain cases the features are not exactly the same but the differences among the applications are minor and, with a slight upgrade, it would be possible to make a single application to cover all the functionalities of the several applications, which would eliminate the necessity of the company owning several legacy systems that are permanently being upgraded.

This is the case of the management of audits and the decision to adopt a unique software solution for all sites worldwide was made. The main benefits of this decision are:

- Considerable cost reduction in software maintenance.
- Reduction of costs and people in functional and technical support to the applications.
- Increase of cooperation and interoperability between sites.
- Achievement of a more robust and complete application thanks to the increase of the number of users and the consequent increase of bug corrections and features' development.

For the previously referred reasons it is evident that the company will gain if there is only one application to do the management of audits. Nevertheless, it is important to explain why Audit Management application will not be used as it is and why it will go through a process of reengineering.

As it was said before, Audit Management was only used in one site of the company and it was specifically developed to be used in that site. Even if the auditing process is very similar in all sites, there are some particularities in the audit workflow that depend on the site where they

are performed. Due to the enlargement of usage to other sites, the application has to evolve to be able to provide features that cover the processing of audits in as many sites as possible.

Moreover, Audit Management is built with obsolete technologies, which compromise the application maintenance and upgrade. Inclusively, some of these technologies are no longer supported by the company that owns them. The architecture of the application is also a barrier to the application maintenance due to the absence of modularity and to the dependence of several systems.

The increase of the number of users also implied new non-functional requirements like flexibility, performance or the ease of installation that are not present in the current application.

The main objective of this project is to obtain a software solution that implements all the functional and non-functional requirements, emphasizing the coverage of all types of audits performed in the company sites independently of their type or workflow. Moreover, it aims at investigating possible solutions to upgrade the application, making it more robust and complete including new features directly related to the management of audits.

1.4 Structure of the dissertation

This dissertation is divided in chapters and starts with this introduction where is made a presentation about the purpose and the context of this project.

In the second chapter is presented the state of the art in the scope of this project. The concepts, the recommendations from some studies done in this area and some software solutions are described in this chapter.

The third chapter consists of describing the problem that motivated the execution of this project and includes an analysis to the software application that has to be reengineered.

The chapter 4 contains the description of the solution found to solve the problem identified in the previous chapter. The solution is divided in two parts: the specification of the requirements of the new application and the analysis and definition of the architecture for the new application.

The chapter 5 is to describe the steps of the implementation of the application and to present the results of the implementation through some screens of the application.

The sixth chapter contains the documentation of three analysis made with the aim to evolve the application of management of audits. This chapter contains all the information that fundaments the results obtained at the end of the analysis.

The last chapter contains the conclusions achieved with the execution of the project and opens a perspective for future developments.

Chapter 2

2 State of the art

2.1 Introduction

In this chapter some important concepts in the context of this project are described and the work already done by other people in areas of management of audits and in software reengineering is also analyzed.

2.2 Basic Concepts

2.2.1 Audit

An audit is an inspection or assessment activity that verifies compliance with plans, policies, procedures or values of a company. It consists of an evaluation of an organization, system, process or product and it is performed by competent, objective and unbiased people, known as auditors. The purpose of an audit is to verify if the subject of the audit operates according to the approved and accepted standards, statutes, regulations or practices. It also helps to determine if it will be possible to maintain the conformity indicators in the future. Auditing is a part of some quality control certifications such as ISO 9000.

Generally, audits' execution is based on a checklist that contains items that should be verified and, at the end, its results are valued. The date and time of an audit are also planned and, as a consequence of an audit, corrective or preventive actions might be performed.

In the context of this project, the considered audits are quality audits which represent an useful tool inside the quality management system because they provide important feedback about the results achieved by the quality policies. There are different types of audits:

- Process Audit – to examine company processes.
- Functional Audit – audit of a specific knowledge area.
- General Audit – any internal audit performed by people of the quality department that doesn't fit in any of the other types.
- External Audit – audits performed by the clients of the company.
- Line Audit – audits to production line.

People involved in an audit may have one of three roles:

- Auditor – person or team that requests and executes the audit.
- Auditee – person responsible for the area, section or process that is audited.
- Action owner – person responsible to perform some action to correct some non-compliance detected during audit execution.

2.2.2 Legacy System

The legacy system is a fundamental concept in software engineering. It is also very familiar to all the enterprise's environments due to the quick evolution of technology.

According to Ian Sommerville [Som00], a researcher on software engineering and author of a popular student textbook in this area, a legacy system is:

"Older software system that remains vital to an organization."

He also refers that:

- *"Software systems that are developed specially for an organization have a long lifetime."*
- *"Many software systems that are still in use were developed many years ago using technologies that are now obsolete."*
- *"These systems are still business critical that is, they are essential for the normal functioning of the business."*

Many authors have written about this issue and have identified some crucial phases and factors to be taken into account throughout the process of replacement of these systems.

There are several ways to replace a legacy application and at the time of choosing one the quality of the application and its business value have to be taken into account. The possible decisions to adopt are:

- Scrap the legacy system and modify the business process if it has low quality and low business value.
- Reengineer or replace the legacy system by a suitable system if it still has a high business value but it is expensive to maintain.
- Replace with COTS[4] if the application has high quality but low business value.
- Continue in operation using normal system maintenance.

The replacement of a legacy application is a risky process because the systems rarely have a fully documented specification and may embed business rules that were not formally documented elsewhere. It is also, in most cases, an expensive process due to the following reasons:

- Different parts implemented by different teams that work separately and don't follow any consistent programming style.
- The system may use an obsolete programming language.
- The system's documentation is often outdated.
- The system structure may be corrupted by many years of maintenance.
- Techniques to save space or increase speed at the expense of the software's understandability may have been used.

[4] Commercial Off-The-Shelf. It is the designation done to commercial software that is ready to sell, lease or license and was developed without any specified client.

- File structures used may be incompatible.

Audit Management fits perfectly in the definition of legacy system given by Ian Sommerville and several reasons listed above were present during the project's execution like the obsolete programming language and the absence of technical documentation.

2.2.3 Software Reengineering

This project consists of reengineering a software solution, therefore, it is important to investigate what was already done in this area and what are the suggestions of the people who have already implemented this type of project. The previously quoted Ian Sommerville [Som00] considers that software reengineering consists of:

"Reorganizing and modifying existing software systems to make them more maintainable."

This quotation sets forth all the possible motives to reengineer a software system because maintainability is the key factor that leads to the replacement of legacy systems. It may, possibly, lead to the erroneous idea that reengineering is part of the application's maintenance. It is not so, because if great changes are done in the application it is no longer maintenance but reengineering. Generally, the frontier between maintenance and reengineering lies on the changes made in the application architecture but it is very flexible and depends on the dimension of those changes.

Based on the analysis made by Dr. Linda Rosenberg [Ros96], reengineering is the examination, analysis and alteration of any existing software system to reconstitute it and its subsequent implementation in a new form. The goal of reengineering is to understand the existing software (specification, design, implementation) and then to re-implement it to improve the system's functionality, performance, implementation and maintainability. Thus, the objective of reengineering is to maintain the existing functionality and prepare it for any functionality to be added later.

Generally, all the reengineering projects have some important and common phases which have to be handled carefully. These phases are:

- Reverse engineering; consists of analyzing existing software with the purpose of understanding its design and specification.
- Re-documentation; consists of creating or updating the documentation of the software system.
- Program structure improvement; as excessive maintenance tends to corrupt the structure of a program, it has to be restructured.
- Program modularization; consists of reorganizing a program to group related parts in the same module.
- Data Reengineering; consists of analyzing and reorganizing the data structures in a program.

The figure below shows a general model for software reengineering.

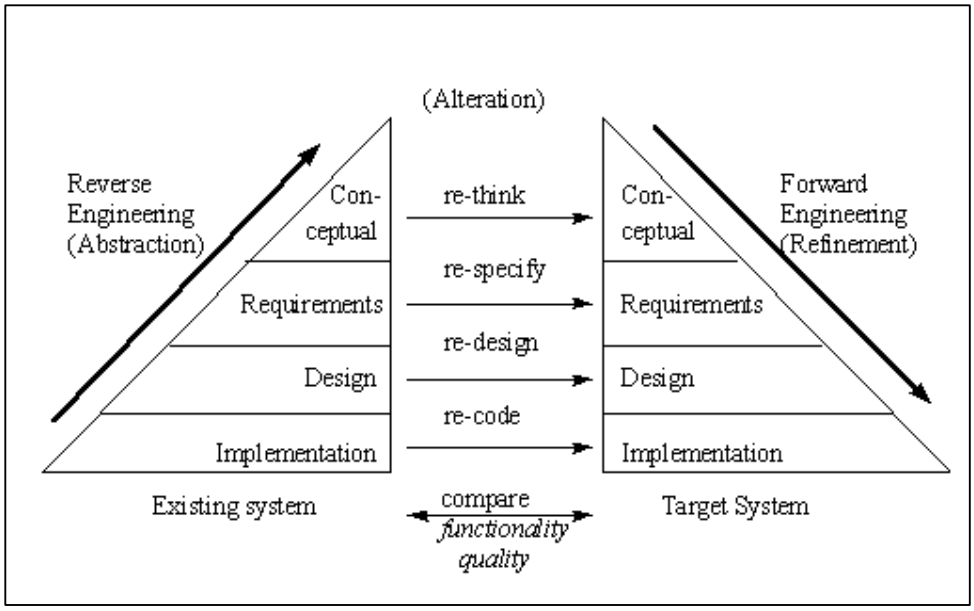


Figure 1 - General model for software reengineering

Software reengineering has both advantages and risks which are very specific to each type of project and have to be taken into account at the moment of deciding to implement a reengineering project. In comparison to the development of a new software system, reengineering has less risks and costs. In the development of new software there is a bigger probability of existing development, staffing and specification problems. With respect to costs, developing new software is more expensive than reengineering an existing one.

The reengineering projects are often executed to mitigate risks but that doesn't mean that the risk is non-existent. Early risk identification is essential for effective risk assessment, risk analysis and risk management. The most frequent risks of software reengineering are:

- Extremely high manual reengineering costs.
- Costs benefits not obtained in required time frame.
- Reengineering effort drifts.
- Lack of metrics program.
- Reengineering with no local application experts available.
- Masses of expensive documentation produced.
- Reengineering technology inadequate to achieve reengineering goals.
- Legacy functionality becomes obsolete prior to reengineering completion.
- Difficult to capture much design and few requirements from code.
- Existing business knowledge embedded in source code is lost.
- Difficulty in migrating existing data.
- Less effectiveness from the programmer when working with an unpopular reengineering project.

- Premature commitment to a reengineering solution of an entire system.
- No plan for using the existing reengineering tools.

There are several approaches for the reengineering of a software system depending on the depth of changes that will be made. It may be a simple code migration or a complete restructuring with architectural changes. The figure below shows the different approaches to software reengineering placed from the least to the most expensive.

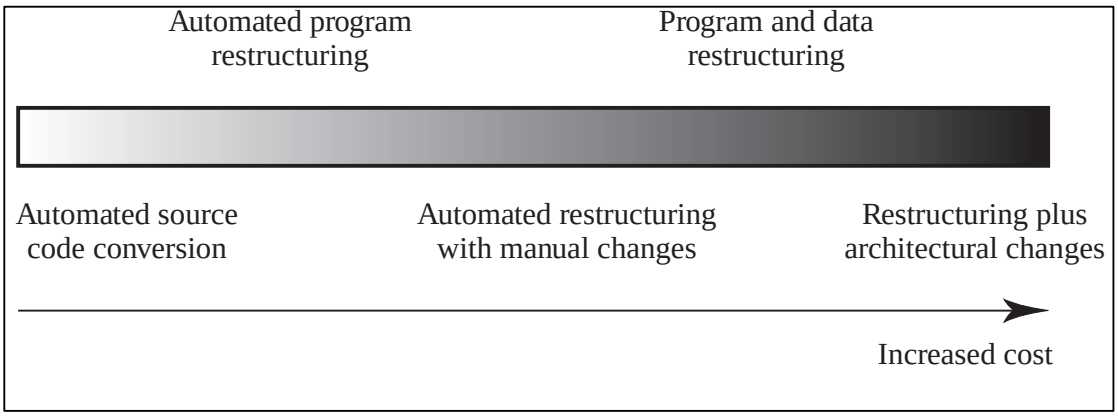


Figure 2 - Software reengineering approaches

The reengineering of Audit Management fits in the most expensive approach because it implies the restructuration of the program and changes in architecture.

2.3 Technologies and tools for software reengineering

The projects of software reengineering often imply the upgrade of technologies and languages of the software system. In the majority of the cases, it only consists of upgrading to a new version of the same technology or to a language considered as the successor of the old language.

The projects where a migration of technologies exists are very frequent in companies, so, there are many reports in the internet of people who have already implemented this type of project. The experiences reported by those people are very useful to reduce risks of reengineering of Audit Management because it is possible to anticipate eventual problems that may happen.

These types of projects were already done inside the company, which made it possible to take notes about the outcome of these projects.

Depending on the type of the reengineering project, it may be useful to have a tool that helps in the migration of the source code of the software system. The tools used to convert the code are especially designated to projects that only need a source code conversion without any concern of improving the system's structure or maintainability. The usage of code converters damages the code readability and, in certain cases, makes it unreadable.

In the concrete case of this project, Audit Management application was built using old Microsoft technologies (ASP and Visual Basic 6) whose support was discontinued and that

application now has to be reengineered to the .NET framework and C# language. Microsoft provides in MSDN[5] some articles and tools to guide developers who will migrate their obsolete applications to the new framework. Apart from Microsoft, there are also several tools that aim at converting a source code from one language to another. Those tools have proven to be useless to this project because the goal of this project is to make a complete reengineering of the software system and not only a simple translation of source code.

For the migration of the source code, in this project, two situations have to be considered. The first one consists of migrating web pages built in ASP code to ASP.NET and the second consists of migrating the source code of some libraries (.dll) written in Visual Basic 6 to C# code. For both cases some solutions that could reduce the development time of the project were analyzed.

- **ASP to ASP.NET migration**

  Among all the solutions investigated and tested only one seems to be useful to convert ASP code to ASP.NET. The application's name is asp2aspx and there is a trial version available in http://www.netcoole.com/download.htm. This application has a trial period of 30 days and converts files with a maximum of 300 lines of code. asp2aspx was tested with some files of the code of Audit Management. If results of the tests had been positive, the purchase of the complete version would have been suggested.

  This application intends to receive as input an ASP solution and to convert it into an ASP.NET solution and it also allows the configuration of some conversion details like the ones shown in the figure below.

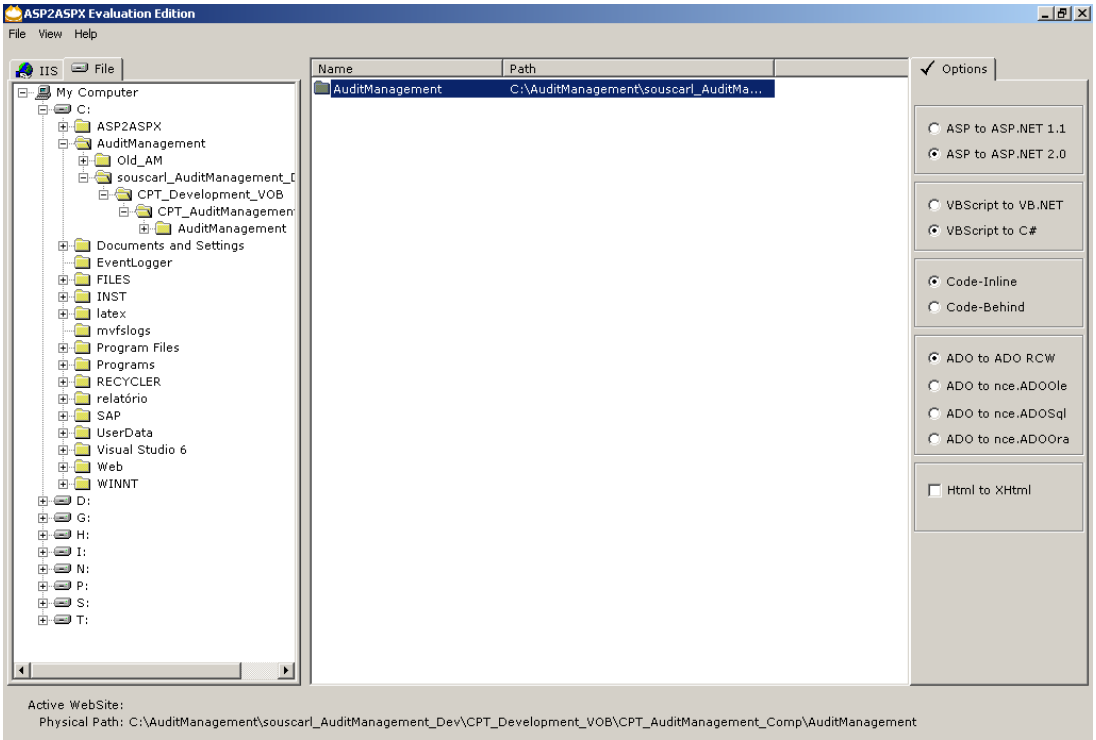


Figure 3 - Asp2aspx user interface

[5] Microsoft Developer Network – Branch of Microsoft responsible for managing the firm's relationship with developers. The web pages of MSDN contain all the documentation of Microsoft languages still supported.

In figure 4 the content of the right menu, where the options for the conversion are, can be seen more clearly.

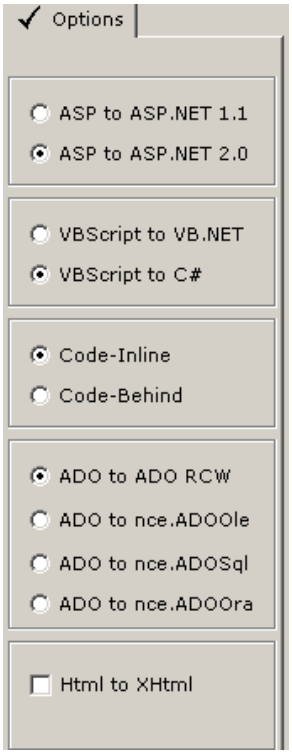


Figure 4 - Options for conversion

The results achieved with this application immediately discard this option because it will increase the time of development instead of decreasing it. The source code generated by the asp2aspx application has low quality because it is unreadable, weakly typed, has some syntactic and semantic errors and is not correctly separated into code-behind.

Besides that, the test of this application helped to prove that the attempt of the translation of the code doesn't fit in the goals of the project. The translation of the source code reduces the maintainability which is one of the most important requirements of the project.

- **Visual Basic 6 to C# migration**

As Audit Management has some library files implemented in Visual Basic 6 language, an analysis was made to verify if there is any way to reduce the time of the development. Two possibilities to do this migration were found. The first one consists of using a tool that translates Visual Basic code directly to C# code. The second possibility consists of translating Visual Basic code into Visual Basic .Net, afterwards Visual Basic .NET code has to be compiled to obtain the .NET assembly file and, finally, the reverse engineering of the generated assembly to obtain C# code has to be done.

The reverse engineering, in this case also known as file disassembling, is possible due to the architecture of .NET framework because its the languages are compiled into an intermediary language as shown in the figure below.

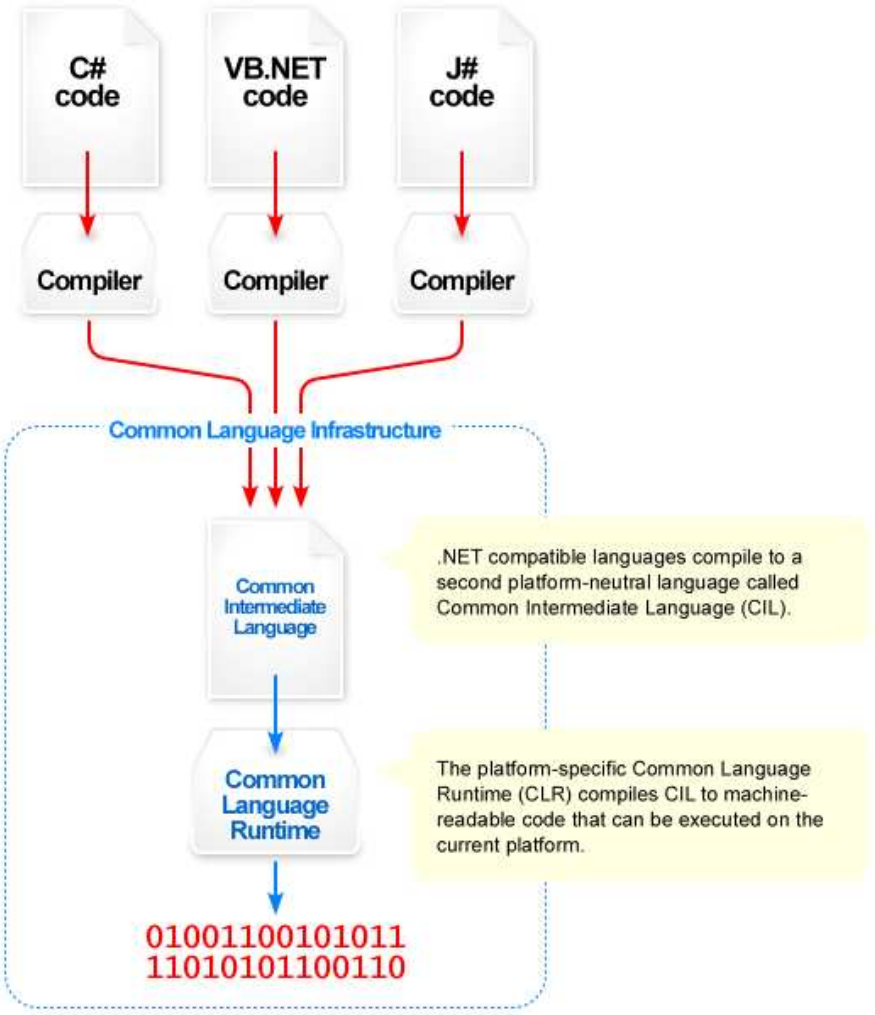


Figure 5 – Steps of compilation in .NET framework

To accomplish the file disassembling a free software utility was used, .NET Reflector, which has features that allow class browsing, static analysis and decompiling. All the features are not available in the standard version but there are some add-ins that may be installed to include additional features. To disassemble a .NET assembly the add-in FileDisassembler has to be installed.

The reverse engineering of the assembly files is only possible before the obfuscation of MSIL – Microsoft Intermediate Language – code. The obfuscation of the code consists of rendering the code unreadable and incomprehensible, to avoid its reverse engineering and prevent the access to the source code. As it is shown in the figure below the .NET framework first compiles the source code into a clean MSIL code, which is easy to decompile, entering, afterwards, obfuscation in the code.

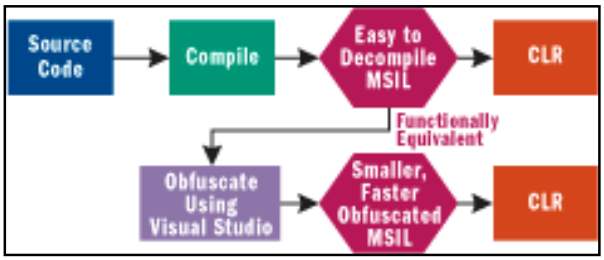


Figure 6 - Obfuscation process in .NET framework

This approach for code conversion was considered because it is probable that a conversion from Visual Basic 6 to Visual Basic .NET will produce better results than a conversion directly to C#. In order to have an idea about the results achieved with this method, some source code files written in Visual Basic .NET were compiled into an assembly file and, afterwards, this assembly was decompiled to C# language. The C# code obtained was neither easy to read nor to understand and the original data types were not maintained. For example, a variable with string type was converted to a variable with object type.

After these two tests of conversion of the code, ASP to ASP .NET and Visual Basic 6 to C#, it was proven that migration of the source code is not profitable for this reengineering project. It could be an useful tool for speeding up other types of reengineering projects but in this case it doesn't fit in the project's requirements, particularly with respect to the improvement of the application maintainability.

The tools previously referred to may be classified as CASE[6] tools due to the usage that was made of them in this project. There are other CASE tools useful for different phases of the reengineering projects, particularly to help in the reverse engineering. This project implies a strong effort in this phase because it is necessary to understand the requirements and the specification of the old application without any documentation. Thus, the source code and the usage of the application were the only means available to understand the program functionality.

A tool that generates flowchart diagrams based in Visual Basic source code was tested. The application's name is Code Visual to Flowchart and it is available in http://www.fatesoft.com/s2f/. This tool was not used because it only generates flowchart for one source file at each time and doesn't allow a general overview of the entire solution.

[6] Computer-aided software engineering – refers to tools that assist in the development of software engineering projects.

After analysis of several tools, it was concluded that the best way to understand the application's requirements and its architecture is the support of an IDE[7]. The IDE programs used were Microsoft Visual Studio and Microsoft Visual Basic 6.0. The best way to identify the features of the application consists of using it as a regular user, who browses all the functionalities provided by it.

2.4 Software solutions for management of audits

An audit is a worldwide concept existing in the entrepreneur environment, particularly in big companies whose quality objectives are almost the same everywhere. Because audits are performed all over the world, many studies were done and several software solutions were developed to support audits. However, each company customizes audits as it sees fit and defines specific audit workflows and types of audits. Thus, it is not easy to have a standard software solution that covers all these particularities.

In the market, there are several companies that have developed software solutions for the management of audits, some of which were considered viable options to replace the current Audit Management application. Internally, Qimonda also owns more solutions for the management of audits that are spread its different sites. Some possible solutions for the management of audits within the company are described in the following sections.

2.4.1 SAP Audit Management

SAP Audit Management, also called SAP AUD, is an application provided by the SAP company in its wide range of software solutions. This is a very complete application with a large set of features and has a high reliability according with the SAP patterns of quality.

Qimonda has a license for all the SAP products, so, the SAP AUD is a possibility to support the management of audits within the company. Due to this, a complete analysis of the advantages and disadvantages of adopting that solution was conducted in the final part of the project. The analysis performed and the conclusions about adopting SAP AUD are described in chapter 6.

2.4.2 Klusa

Klusa is a large software application with several purposes currently used at the Qimonda site in Porto. Klusa is subdivided into functional modules, offering features to support several processes of medium or large companies.

The main functions provided by Klusa consist of multi-project management, demand management and quality management. The management of audits is included in the quality management function.

This application is, in fact, at the origin of the decision of reengineering Audit Management application because the usage of Klusa will be discontinued. This application represents a high annual cost for Qimonda and, for that reason, the decision was made to terminate the contract with the company that owns the application.

[7] Integrated Development Environment – software application that provides comprehensive features for software development.

2.4.3 Casanova

Casanova is a software solution that belongs to Qimonda, so, doesn't have licensing costs, and its purpose is to support audits. Nevertheless, it only allows the management of self-assessments in the Qimonda environment. The self-assessment is an efficient tool to determine the strengths of the company's processes and the areas to improve according to the PDCA[8] sequence.

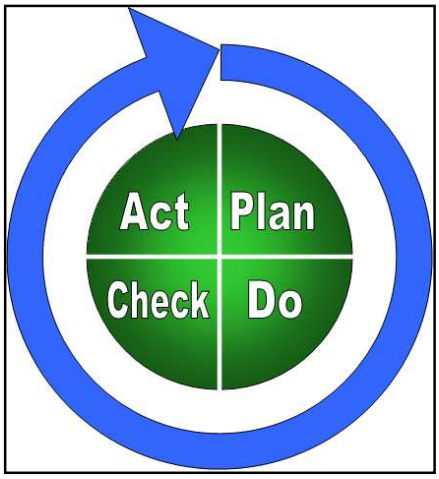


Figure 7 – PDCA sequence

The self-assessment is similar to an internal audit because its goal is to assess the processes performed within the company. The difference between these two inspection mechanisms is that internal audits are executed by people of the quality department while self-assessments are executed by people responsible for the process assessed.

The existence of these two mechanisms is indispensable to guarantee the quality in all the internal processes. Internal audits confer neutrality to the assessment because they are executed by people unfamiliar to the process and during self-assessments, generally, more mistakes are found, which complies with H. J. Harrington's quotation:

"No one knows better if someone is doing their job correctly than the person himself."

Although it is possible to use this application for the management of self-assessments and the management of internal audits due to their similarity, the application doesn't fit totality with all the existing audit requirements. For example, Casanova doesn't support external audits or general audits, so, it can't be adopted as it is, to do the management of audits.

As Casanova would require a high development effort of new features and Qimonda owns other applications that require a lower development effort, this option has been excluded from the possible set of solutions for the management of audits.

[8] Plan-Do-Check-Act – iterative four-step problem-solving process typically used in quality control (in Wikipedia).

2.4.4 Gestão de Auditorias

Gestão de Auditorias is a software application entirely designated to the management of audits and it belongs to a Portuguese company specialized in the development of software for quality management. This application appears as an option for the management of audits because Qimonda has recently purchased an application to that company for other function in the quality department and, only then did the company know of the existence of the application *Gestão de Auditorias.*

This application is web-based and is hosted in the servers of the company seller, so, if it is purchased, it will not be necessary to install anything in the client's computers or in the client's servers.

The advantage of this solution is its low cost in comparison to other solutions available in the market. Moreover, it doesn't bring any new cost to the company concerning hardware maintenance or disk space in the servers but, as Qimonda owns several solutions for the management of audits, the usage of one of them is recommended. Thus, this solution will only be considered if it is proven that no other application meets the specified requirements.

2.5 Conclusions

The analysis of the state of the art helped in the planning of the project. Some guidelines and recommendations were followed according to the study of the state of the art.

The projects of software reengineering were object of several studies that identified common phases to all the projects of this type. Due to that, several authors have written about the best practices that have to be followed in those projects and the most frequent risks. Thus, before the start of the project, the recommendations were taken into account and some measures were taken to reduce the risks of the project's execution. For example, as it is frequent that the legacy systems contain rules embed in source code that are not present in the documentation, the analysis of the source code of the Audit Management application was more intensive to not avoid some of those rules.

The analysis of the different approaches existing to reengineering projects allows concluding that the reengineering process of this application can't be done with several automated tools. The usage of those tools doesn't fit several requirements of the project including one of the most important of them which is the flexibility of the source code.

Some software solutions for the management of audits existing in the company or in the market were analyzed and leads to the conclusion that the best solutions for the management of audits at Qimonda are Audit Management and SAP AUD. Inclusively, in the final phase of the project, a comparative study between these applications was made.

Chapter 3

3 Application for management of audits

3.1 Context

This chapter aims at presenting the main reasons that lead to the decision of this project implementation and the objectives that the project intends to achieve.

3.1.1 Purpose of the product

Audit Management application, a software tool used by the Qimonda quality department, replaces the manual way of tracking the audit process. This application keeps a record of all the relevant information about the audits, and provides online auto reporting improving, thus, the visibility and control of the audit actions. It also helps to ensure responses within a time frame to actions to be put in place thanks to an email reminder mechanism.

3.1.2 Areas of application, system integration and delimitation

Audit Management application is currently in productive usage at the Qimonda site in Malacca (Malaysia) and, at short term, it will be used at the Qimonda sites in Porto (Portugal) and Suzhou (China). This application helps the people of the quality department to maintain an electronic record of audits performed within the company and also allows the monitoring of audits in progress.

The integration with some existing systems in the Qimonda environment and the integration with other software solutions was the subject of study during the project development. The prototype implementation allows verifying if the inclusion of this application in the company originates any conflict with other existing software systems. It also allows verifying the compatibility with the hardware configurations.

Each site of the company that wants to use Audit Management needs to install a new instance of the application for its exclusive usage. As the application is available through the intranet of the company it would be possible to have a single installation of it and provide access to all sites worldwide. However, this situation is not possible due to the company policies for the support and maintenance of the applications and also due to the confidentiality of the information treated. Each site is not interested in sharing information related to its audits with other sites.

3.1.3 Users

Audit Management application may have several different types of users, depending on the type of audit. However, most of the users belong to the quality department and have the auditor role in the audit, this is, the person responsible for performing the audit. In the case of external audits which are performed by clients of the company, the person that performs the audit doesn’t interact directly with the application, so, when it comes to entering the data in the application the auditor role is taken by someone of the quality department.

The users of the application may also be from other departments if they have auditee role. The auditee is the responsible for the process, function, material or area that is audited and may belong to any department of the company.

As consequence of an audit, some deviations may be detected, which have to be corrected through the execution of actions. For each action there is a person responsible for its execution. That person is identified in the application as the action owner and may also belong to any of the company's departments. He is designated by the auditee and has to report the action status to him.

There is another type of user that is responsible for the execution of the administration of the operations.

3.2 Reengineering motivation

The application for the management of audits is built with obsolete technologies which can compromise the application maintenance and upgrade. Moreover, even if the basic principle of audits is the same in every Qimonda site, there are some details that depend on the facilities where the audit is performed. Therefore, the application has to be sufficiently flexible to be easily adapted to the particularities of each site. The required flexibility is not present in the current application due to its architecture and technologies, so, it is necessary to upgrade the architecture and technologies to the latest guidelines standards.

To summarize, the main motives that lead to the decision that current Audit Management solution isn't adequate for the management of audits and, consequently, to the execution of this project are:

- Difficult maintenance and upgrade of the application.
- Obsolete technologies which are no longer supported by the company that developed them.
- Poor performance of the application and user complaints concerning the time spent by the application to perform some operations.
- User requests for some changes in the application.
- Architecture's design confers low flexibility to the application.
- The necessity to evolve the application with the integration of new features like the management of action plans.
- Difficulty to install the application.

This point has been included due to the several migrations of servers. Each of the migrations implies a new installation of the application. Besides this, it is also necessary to install it in several sites.

3.3 Reengineering objectives

The objective of the project may be summarized as the resolution of all the problems previously identified. Thus, after the reengineering project, Audit Management should be easy to maintain and upgrade, have better performance, be flexible and easy to install.

Nevertheless, there are some steps that have to be done before the final objectives of the project. All the phases of the software development process have to be reached and, due to the particularity of this project, additional phases have to be included. Thus, the intermediary objectives of the project are the following:

- Identify the requirements for the application based on the old Audit Management application and on the user's requests.
- Analyze the solutions of the company that may be used in the development of the application.
- Redefine the application architecture.
- Define the roadmap of implementation.
- Implement the application.
- Test the application as a concept proof of the architecture definition in terms of productivity and flexibility.
- Analyze the integration of a feature for the management of action plans.
- Analyze mechanisms of reports extraction.
- Evaluate SAP Audit Management solution.

The execution of this project includes the production of software documentation according to the internal guidelines and templates for each phase of the project development. The software development process in the company is based on SEM-Q[9], RUP[10] and Qimonda best practices experience and provides guidelines, templates and tool support. In this project a requirements specification document, a design document and a test plan document were written. During the project's execution a web page was maintained in a Sharepoint[11] site with all the documentation produced and all the relevant information about the project's evolution like the project plan, meeting dates, logbook, contacts, etc…

[9] Systems development method for Infineon.

[10] Rational Unified Process – comprehensive process framework that provides industry-tested practices for software and systems delivery and implementation and effective project management.

[11] Web-based platform for collaboration and document management.

3.4 Audit Management analysis

Before the reengineering project, Audit Management was built as shown in the following figure.

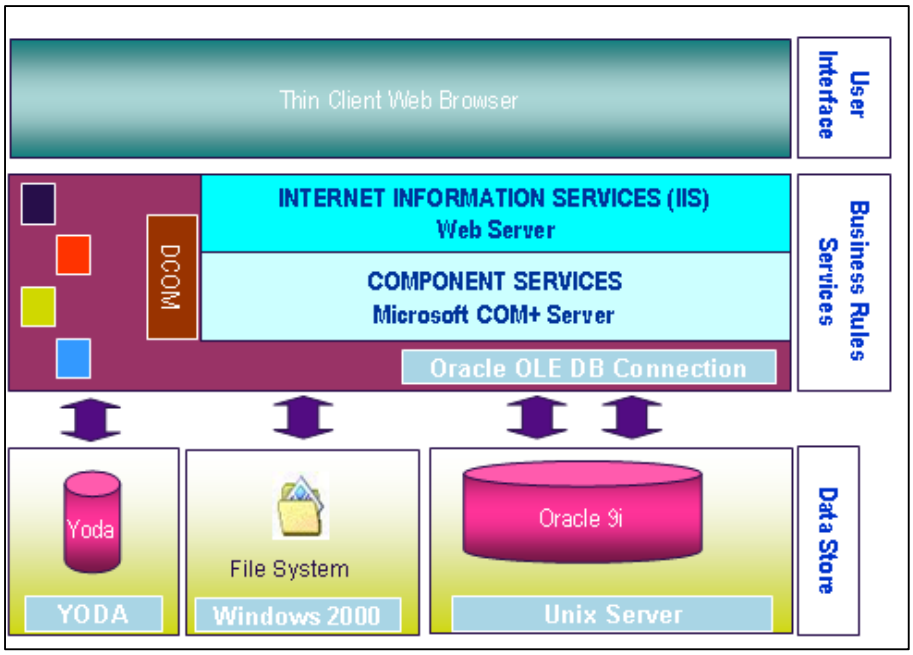


Figure 8 - Audit Management old architecture

3.4.1 Web component

The web component of the application architecture was built in ASP technology using Visual Basic Script and JavaScript as client-side scripting languages. It is used HTML code and CSS for the presentation of the web pages. The most common division of code within a source code file for a web page is as shown below.

Page Directives **User Validation**
Code for Actions (Button click, etc…)
Presentation Code
Code for validation of inputs and user interface actions

Figure 9 - Structure of ASP file

The first layer at the top of the ASP file contains several directives like the identification of the scripting language. It also includes a portion of code responsible for checking if the user who logged in the application has permission to access the web page.

The second layer contains a code to be executed in server-side after some user action. Generally, it is after a click in some button which implies a post back to the server. It is in this section of the file where the code for the data accesses and the execution of database queries is placed.

Below the second layer, the HTML code with ASP code embedded for the presentation of the information is located.

In the bottom layer the code to be executed at the client-side is located. This code is written in the most of cases in JavaScript language nut, in some pages, there is some Visual Basic Script code. This code is responsible for a first validation of user inputs and for some actions in the user interface like the update of dropdown lists or the change of the visibility of some elements.

3.4.2 Business rules

The architecture application doesn't have an effective division of the presentation layer and the business logic layer. Almost all the code for the business logic and the access to data is present in the files of the web component. Nevertheless, the application architecture is composed by COM+ components which are responsible for the generation of reports into Microsoft Excel files.

The implementation of COM+ components starts with the writing of some code in Visual Basic 6 language that was compiled into a DLL file. After that, those files are added through the Component Services application interface to the operating system and are covered by all the properties of the COM+ technology like distributed transactions, resource pooling, better management of memory and processor or event publication and subscription.

3.4.3 Data storage

The application architecture is composed of three systems responsible for storing the data needed by the application.

The main storage system is an Oracle relational database hosted in a Unix server. The access to the database is made using the Microsoft OLE DB Provider for Oracle (MSDAORA)

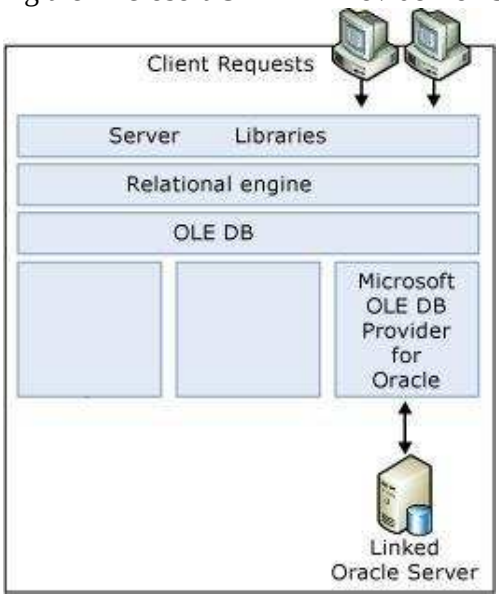


Figure 10 - Layers of database access

The application also interacts with the file system for the storage and consultation of the report files. Some files that may be uploaded by the user of the application to add extra information to the audits are also stored.

The last system with which the application interacts is the YODA system. YODA is a middleware existing in the company based in TIBCO Rendezvous which is software that provides a message bus among enterprise applications. The inclusion of this system in the application architecture is due to the authentication of users. The Audit Management application sends a message to other application through the message bus provided by this middleware, with the username and the password of the user. The application verifies if user credentials are correct.

3.4.4 Physical Architecture

The architecture of Audit Management is physically distributed as shown in the figure below.

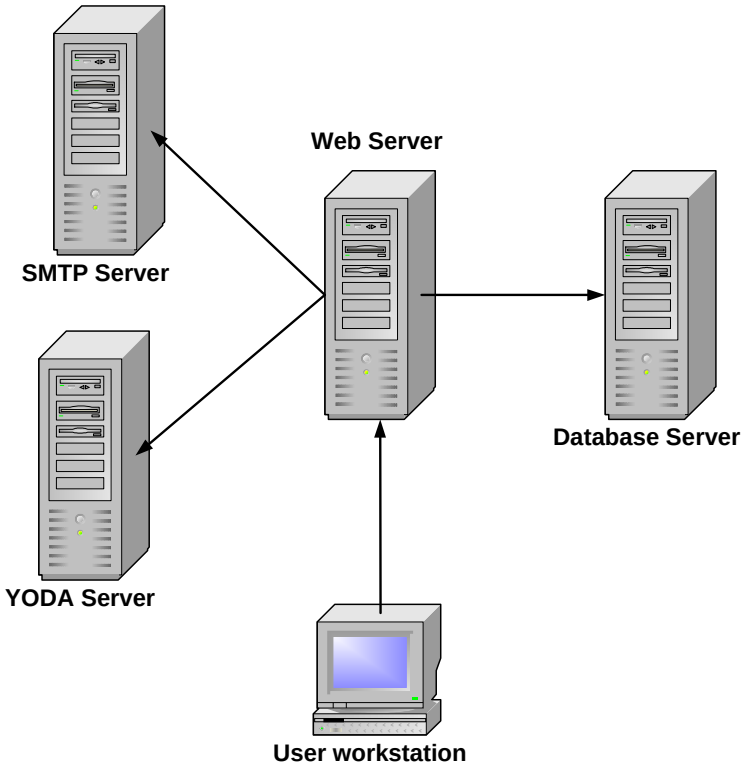


Figure 11 - Old physical architecture of Audit Management

The operating system Windows Server 2003 with the Internet Information Service (IIS) 6.0 which runs the code for the web pages of the application is installed in the web server. The COM+ components responsible for the generation of the reports are also installed in this server.

The YODA server is represented by one server but, in reality, this system is supported by several servers due to the necessity to confer fault tolerance and load balanced to the applications.

The figure below represents a UML deployment diagram, which allows viewing where the logical components of the application are physically located.

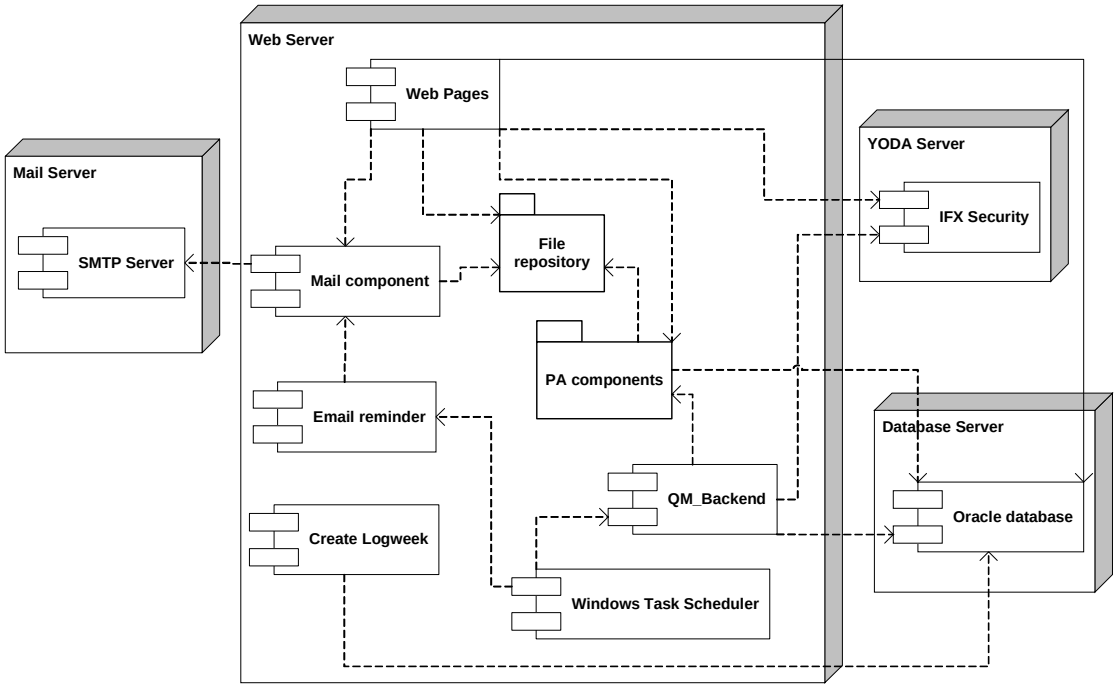


Figure 12 - Deployment diagram of old Audit Management architecture

The less obvious components of the diagram above are:

- **IFX Security:** application that belongs to the YODA middleware used to do the authentication of users in Audit Management.
- **Mail Component:** COM+ component responsible for the sending of emails. The ASP technology doesn't include a feature for email sending, so, it is necessary to install this component to provide this feature to the application. This component interacts directly with the SMTP server.
- **Email reminder:** Executable file that verifies if there is some email that has to be sent and, if there is any email, it invokes the Mail component to the send the email.
- **PA components:** all the COM+ components responsible for the generation of reports. All these components are visible in figure 13.
- **QM_Backend:** Executable file that performs, when executed, some maintenance operations in the file repository (delete old files, update of other files, etc…),
- **Windows Task Scheduler:** The executions of QM_Backend and Email Reminder have to be programmed to run periodically.
- **Create Logweek:** Executable file that updates a table in database with the weeks relative to the fiscal year of the company.

The figure below shows the division of the functionalities of reports generation into the COM+ components. Like it is possible to see there are three modules that provide global functions to all the components.

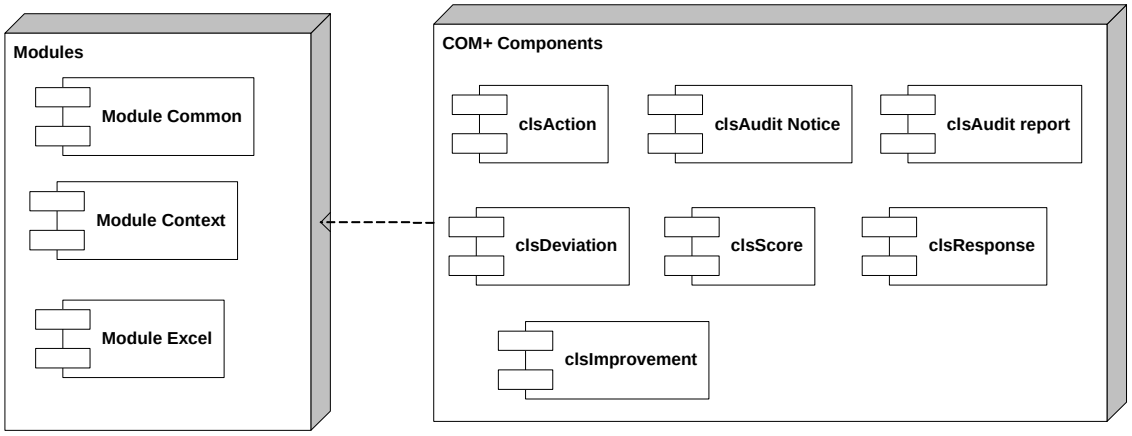


Figure 13 - COM+ components and modules

Chapter 4

4 Specification of the Solution

This chapter explains the solution adopted for the problem described in the previous chapter, namely the reengineering of the Audit management application. This solution was achieved after two main phases: the analysis of the requirements and the definition of the application architecture.

4.1 Requirements analysis and specification

The analysis of requirements aims at identifying the functionalities that will be provided by the application and how those functionalities will be performed as well as the most important non-functional requirements.

The requirements were specified according to the functionalities provided by old Audit Management application to the requests made by the key users of the application and to some discussions with the IT application owner[12].

During the analysis of the requirements of Audit Management, some functionalities were discovered to be useful if included in the application. Nevertheless, the approval of new requirements for the application is a complex process that involves discussion among all the key users worldwide, the IT application Owner and the IT support members. Thus, those new requirements identified as useful will not be implemented so as to not prolong the development time of the application and to quickly obtain a functional application. However, those requirements are documented in the requirements specification document and identified as an opportunity of improvement of the application for a later phase, after the approval by the competent stakeholders.

The functionalities of old Audit Management application have to be maintained after the project reengineering except if there is any indication given by the key users to change or remove some functionality. So, due to the absence of documentation, it was necessary to make a complete study of the application to identify all the functionalities and their business rules.

The specification of the requirements is especially addressed to the developers of the application and it has to provide a complete description of the content, functionality and quality demanded by prospective users. For this reason, the whole document with the requirements specification is available in appendix 1 and, in this chapter, an overview of the requirements is given.

[12] IT application owner is the person responsible for monitoring and control the technological support of the application.

4.1.1 System Flows

The workflow of an audit depends on its category, so, Audit Management application has to support the different workflows. These workflows are described through the UML activity diagrams in following sections.

An audit also has a state which identifies the phase of the workflow where it is. The designation of these states is important because it is used in the application; therefore, a note to identify which state the audit is in was included, in the right column of the activity diagrams.

Independently of its category, an audit always involves three types of users: auditor, auditee and action owner. The interaction of these users with the application is explained in section 4.2.1.1 of this thesis.

4.1.1.1 Process or Functional audits

The activity diagram below represents the flow of usage of the Audit Management application for process or functional audits.

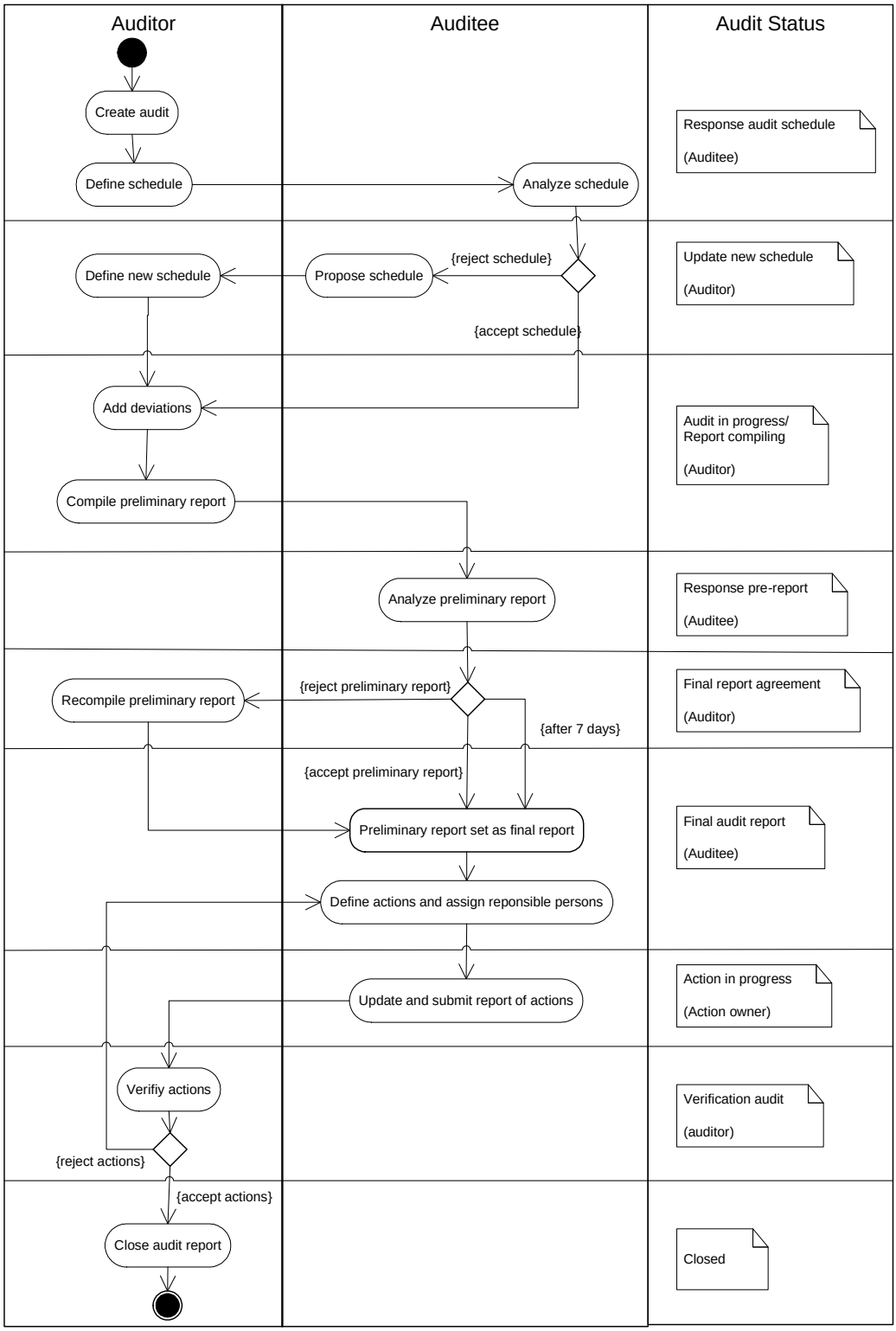


Figure 14 - Activity diagram for process or functional audits

4.1.1.2 General or External audits

The general audits and the external audits have the same flow of usage in the application.

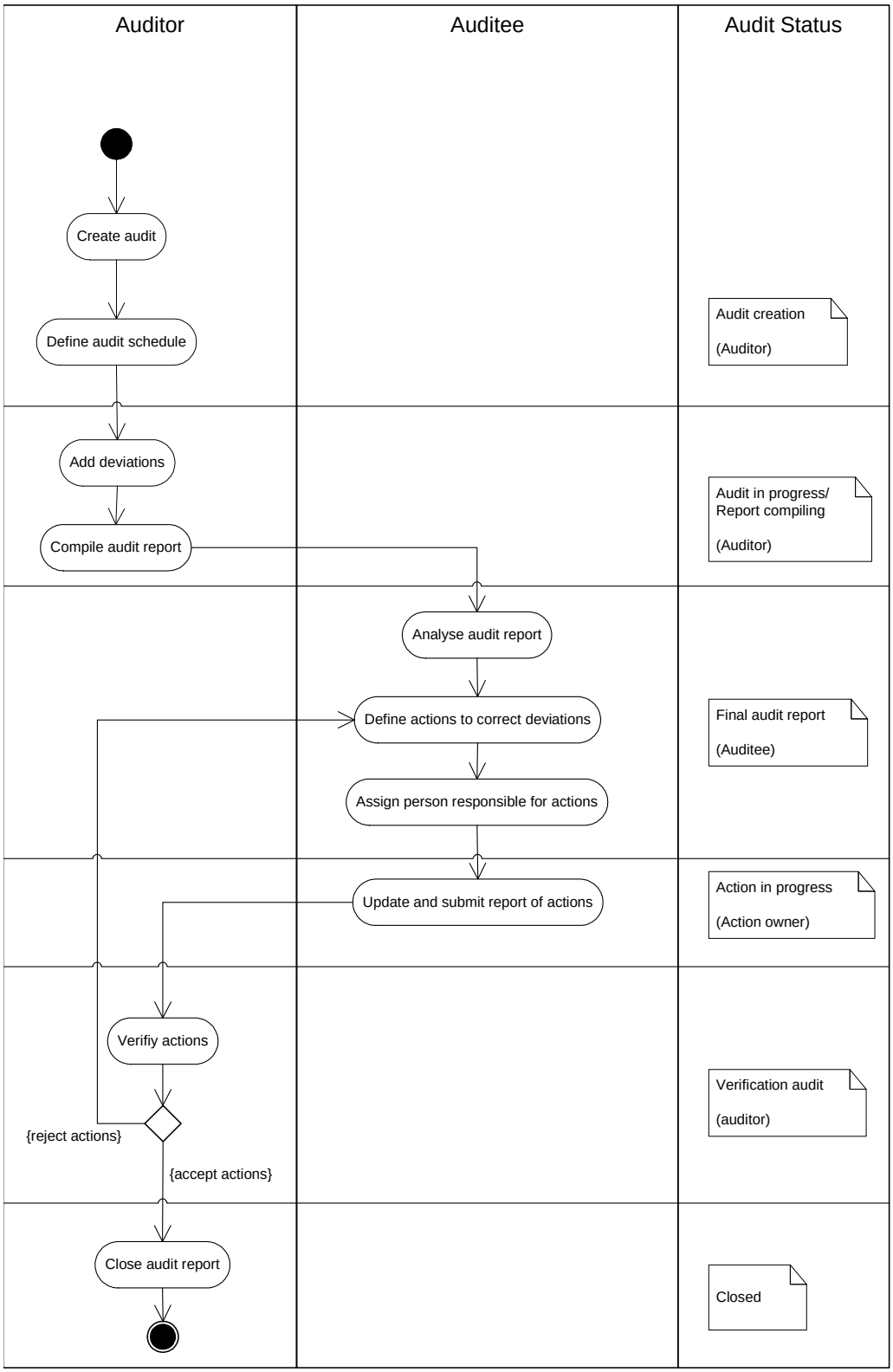


Figure 15 - Activity diagram for general or external audits

4.1.1.3 Line Audits

The audits executed to the production line have a slightly different workflow in relation to other audit types. This workflow is shown in the diagram below.

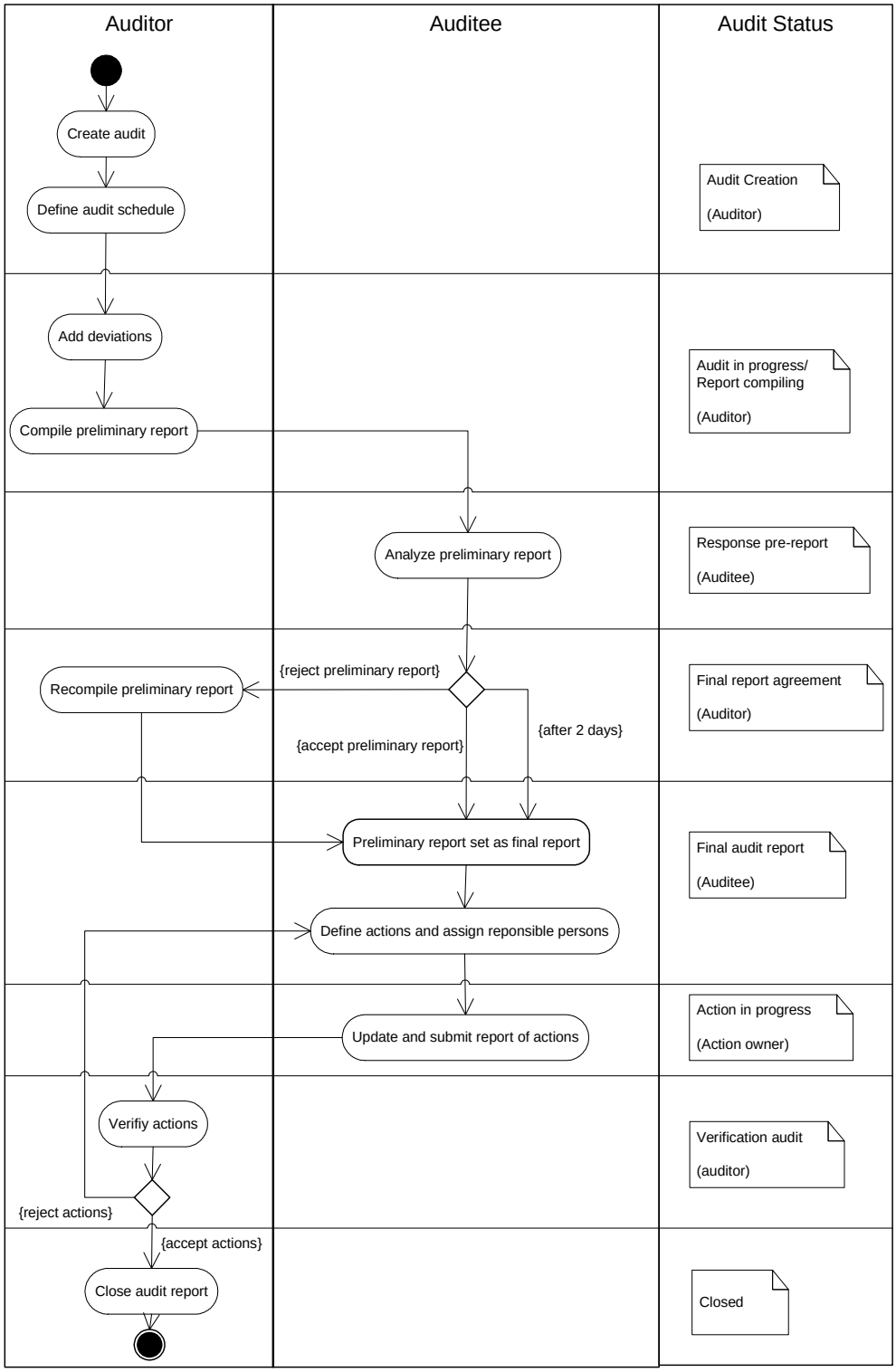


Figure 16 - Activity diagrams for line audits

4.1.1.4 Action States

To correct the deviations detected during the audit execution, it is necessary to define some corrective or preventive actions. The actions are defined by the auditee who also assigned a person to perform the action (Action Owner). After the action completion by the action owner, the auditee has to update the action state in the application. Finally, the results of the action are verified by the auditor who evaluates efficacy of it.

The UML state diagram below illustrates the possible states of an action and the trigger for transition between states.

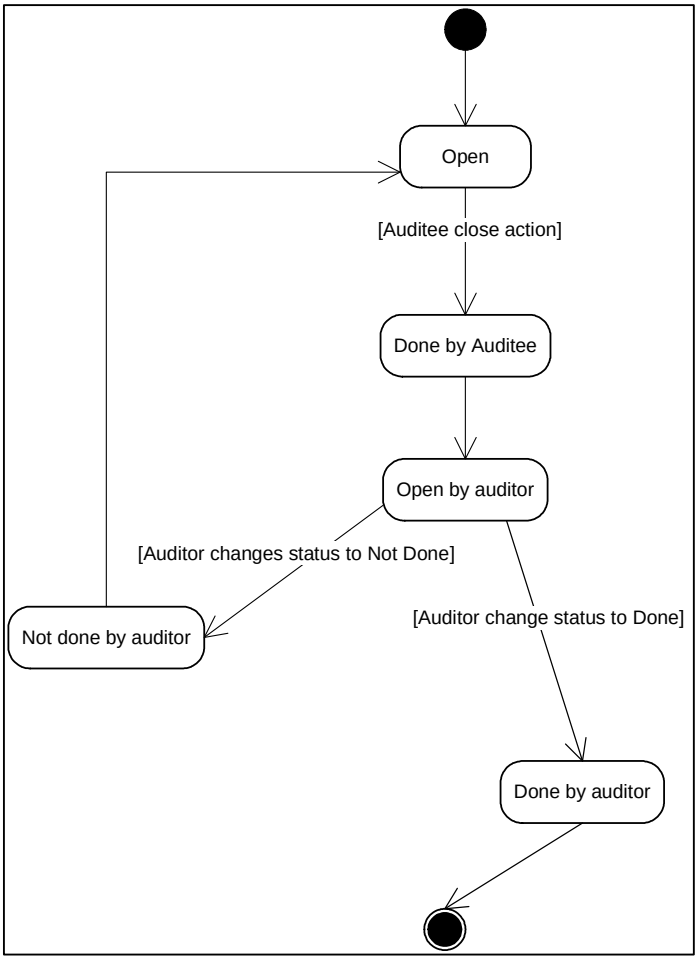


Figure 17 - State diagram corrective or preventive actions

4.1.2 Functional Description

4.1.2.1 Actors

There are 4 types of actors who will interact with the system.

- **Normal user:** this actor has read-only permissions, that is, it can only visit some pages without relevant information about the audits.
- **Auditor:** this user is responsible for the creation and execution of the audit and, in a later phase, he is also responsible to do the follow-up of the actions performed. This follow-up aims to verify if actions have corrected the deviations detected during the audit execution. Out of the context of the application, this user belongs to the quality department of the company and may represent a whole team. In the case of the external audits, these are executed by the clients of the company that don't have access to the application, so, a person in the quality department inserts the data relative to the audit in the application.
- **Auditee:** this user is responsible for the area, system, process, project or product that is audited. The main responsibility of this user in the audit process is to define the actions that have to be performed to correct the deviations detected.
- **Administrator:** this user has all the permissions in the application. Beyond all permissions of the other users he can also configure the application.

The users referred to above, are those who are currently allowed by the application and those who will also be allowed at the end of the project. However, an opportunity of improvement consists in the addition of a new user role corresponding to the Action Owner in an audit. The Action Owner is responsible to perform an action and has to inform the auditee when action is completed. After that, the auditee has to update the action state in the Audit Management application. To avoid this situation, a new user type (Action Owner) that has the permission to update the state of the actions to which it is assigned could be created.

4.1.2.2 Use Cases Overview

The specification of the use cases is an useful tool to identify all the functionalities of the application and to monitor the development of the project. The complete list of use cases is present in the requirements specification document available in appendix 2. In the following sections only the most important ones are described.

The use cases of the application are grouped by packages according to their functionality. In the figure below the packages of the use cases and the users associated to these packages are shown.

The specification of use cases includes the suggestions for the evolution of the application at functionality level. These use cases are distinguished with a yellow color.

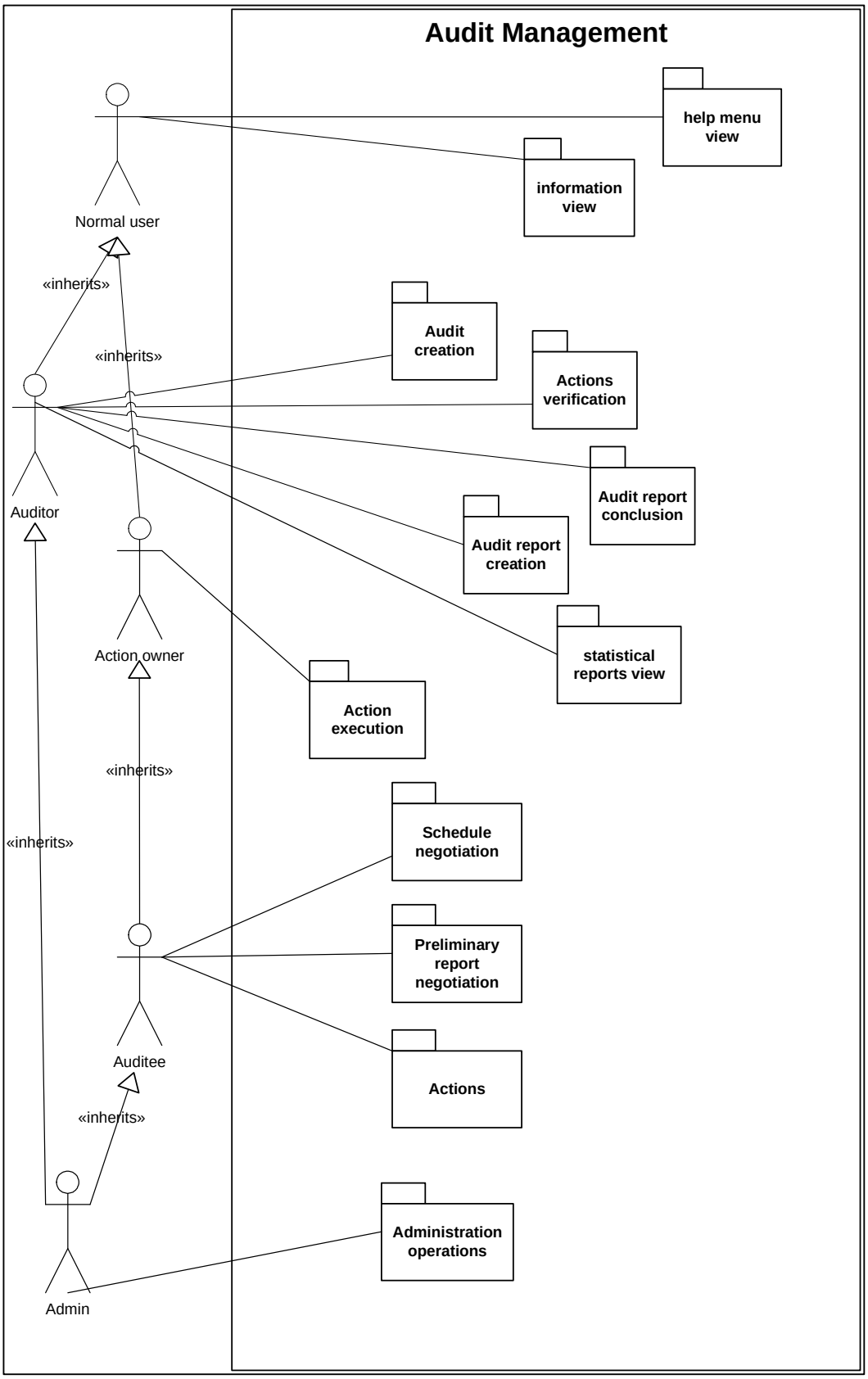


Figure 18 - Use cases packages

4.1.2.3 Package “Help Menu View”

This package contains the use cases correspondent to the help menu of the application.

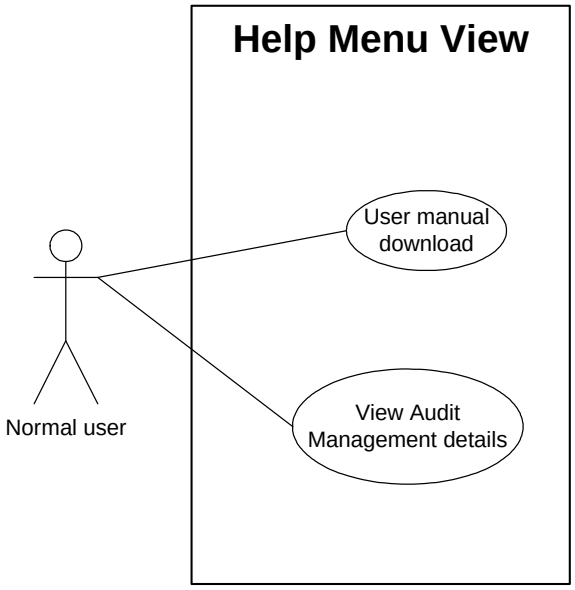


Figure 19 - package "Help Menu View"

4.1.2.4 Package “Information View”

This package contains use cases for viewing information about audits and about the user logged in the application.

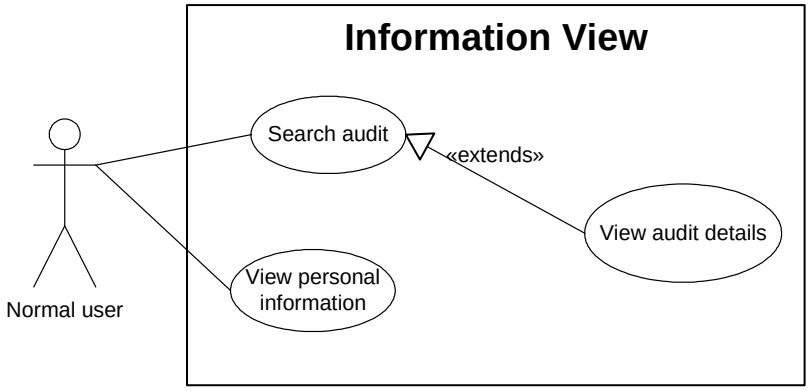


Figure 20 - Package "Information View"

4.1.2.5 Package “Audit Creation”

This package includes all the use cases related to the creation of an audit.

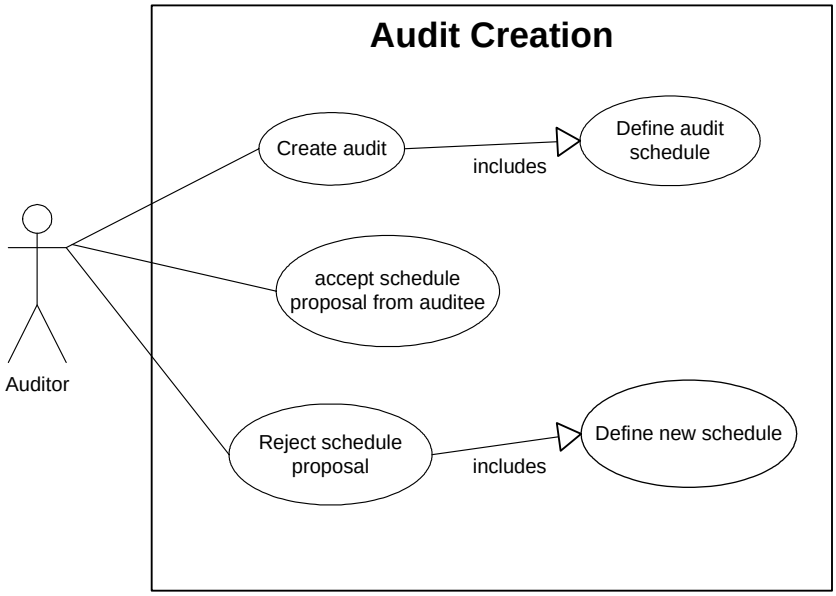


Figure 21 – Package "Audit Creation"

4.1.2.6 Package “Actions Verification”

The use cases of this package are relative to the actions follow-up by the auditor. The use case with yellow background consists of an improvement suggestion that will only be implemented after approval by the competent stakeholders.

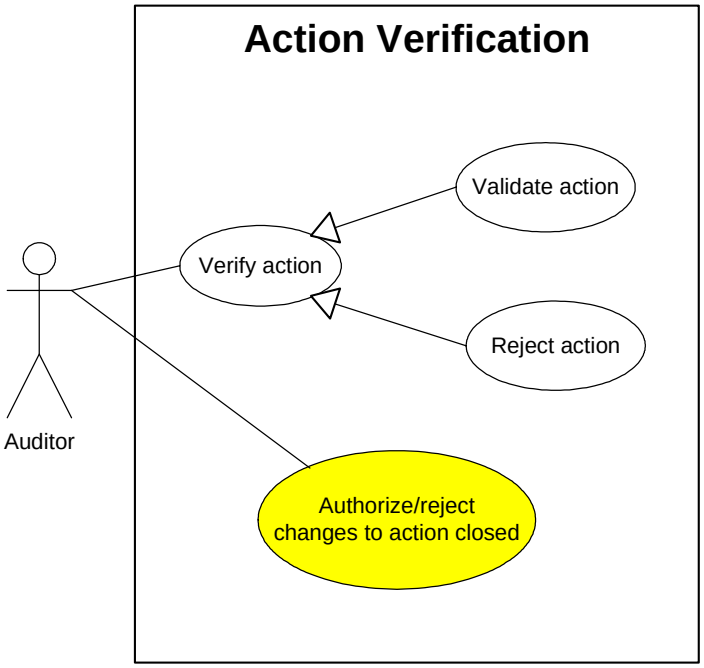


Figure 22 - Package "Actions Verification"

4.1.2.7 Package “Audit Report Conclusion”

This package contains the use case for the conclusion and consequent closure of the audit report.

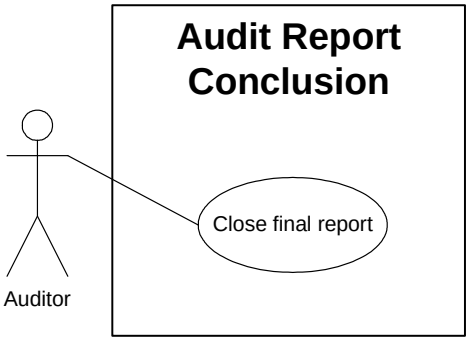


Figure 23 - Audit Report Conclusion

4.1.2.8 Package “Audit Report Creation”

This package contains the use cases related to the creation of the audit report. An audit report consists of a list of all the deviations detected during the audit execution.

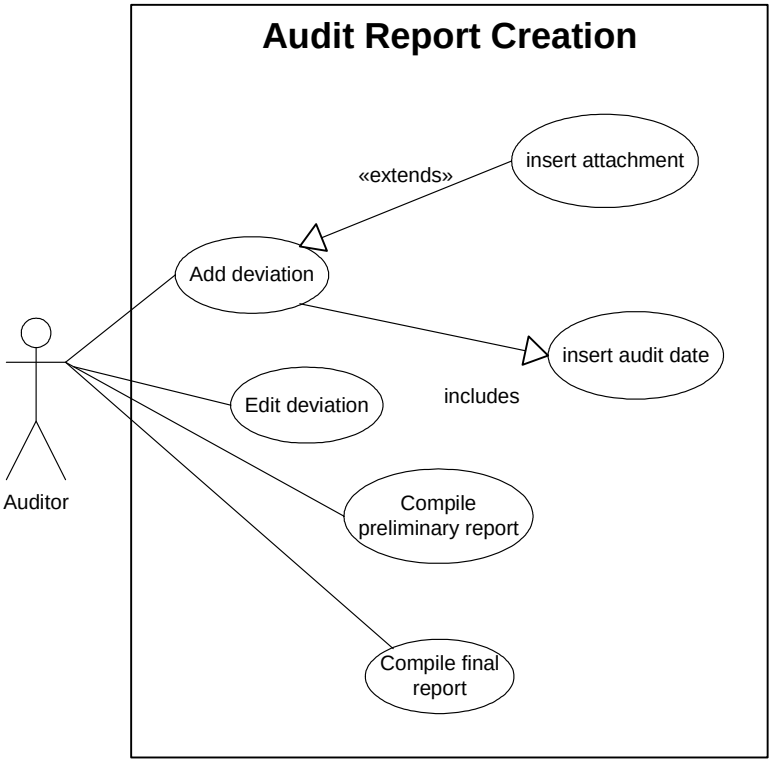


Figure 24 - Package "Audit Report Creation"

4.1.2.9 Package “Statistical Reports View”

The application has to provide some reports containing statistics about the results of the audits executed. These reports have to be stored in the file system and their values have to be updated when user wants.

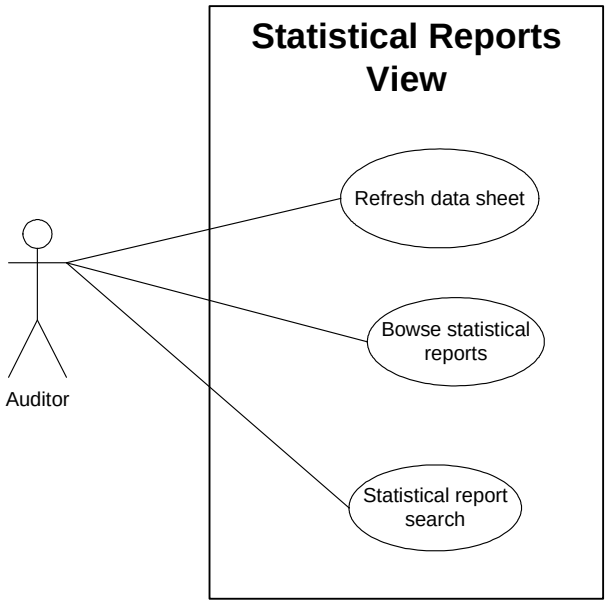


Figure 25 - Package "Statistical Reports View"

- **Refresh Data Sheet:** Use case that allows updating of the data in the reports stored in the file system with the latest values of the audits.
- **Browse Statistical Reports:** Use case that allows the user to browse all the possible statistical reports.
- **Statistical Report Search:** Use case that allows the user to search some report according to the filter specified by him/her.

4.1.2.10 Package “Action Execution”

This package of use cases will not be implemented before its validation by the competent stakeholders but it represents a possibility of evolution of the application. The use cases of this package imply the introduction of a new user role, the Action Owner, which can update the state of the actions executed by him/her.

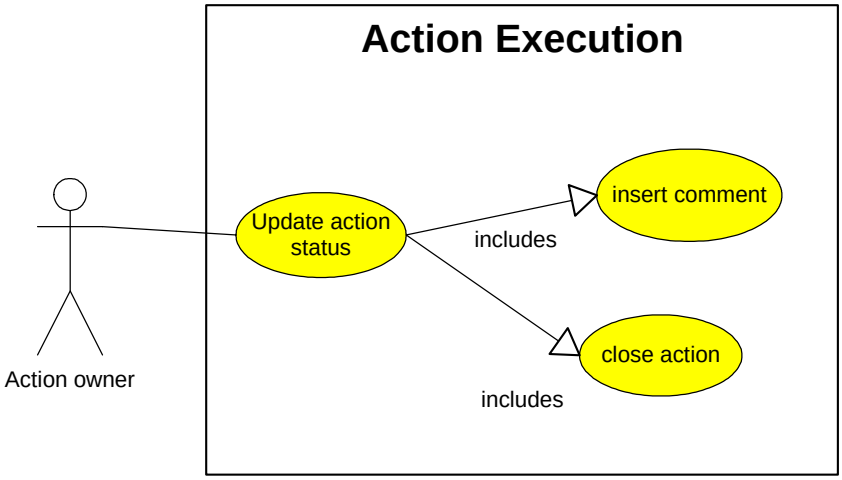


Figure 26 - Package "Audit Execution"

4.1.2.11 Package “Schedule Negotiation”

The use cases in this package are related with the scheduling of the audit on the auditee side. The auditee can accept or reject the schedule proposed by the auditor and, in the case of rejection, he may propose a new audit schedule.

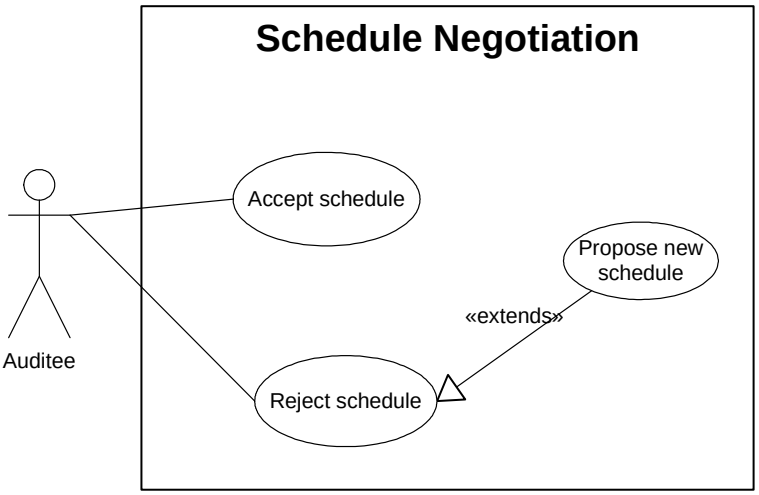


Figure 27 – Schedule Negotiation

4.1.2.12 Package “Preliminary Report Negotiation”

This package contains the use cases for the auditee’s part in the negotiation of the preliminary report. This report is the first version of the audit report and it contains the deviations detected by the auditor during the execution of the audit. The auditee has to review the preliminary report and may agree or not with the deviations identified. If he accepts the deviations, the report is converted to final audit report otherwise the auditor and the auditee have to discuss the deviations and the auditor has to recompile the audit report.

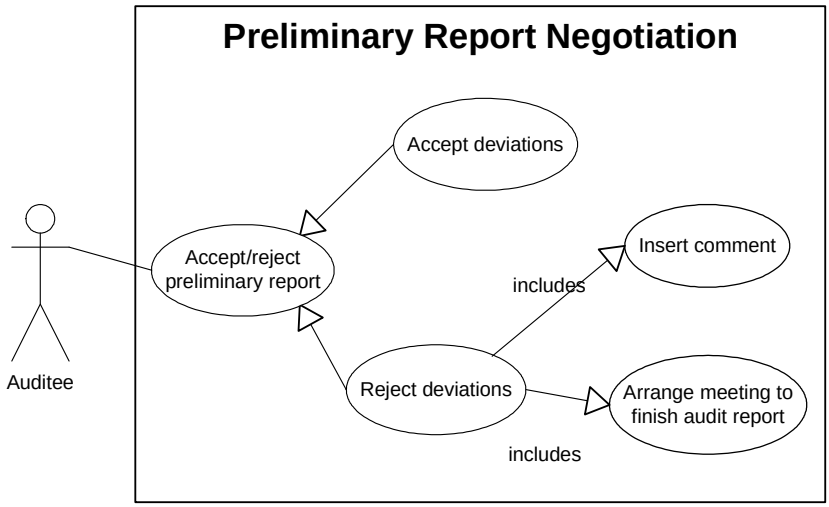


Figure 28 - Package "Preliminary Report Negotiation"

4.1.2.13 Package “Actions”

This package contains the use cases related with the actions’ definition, assignment and execution. The auditee has to define at least one action to each deviation detected.

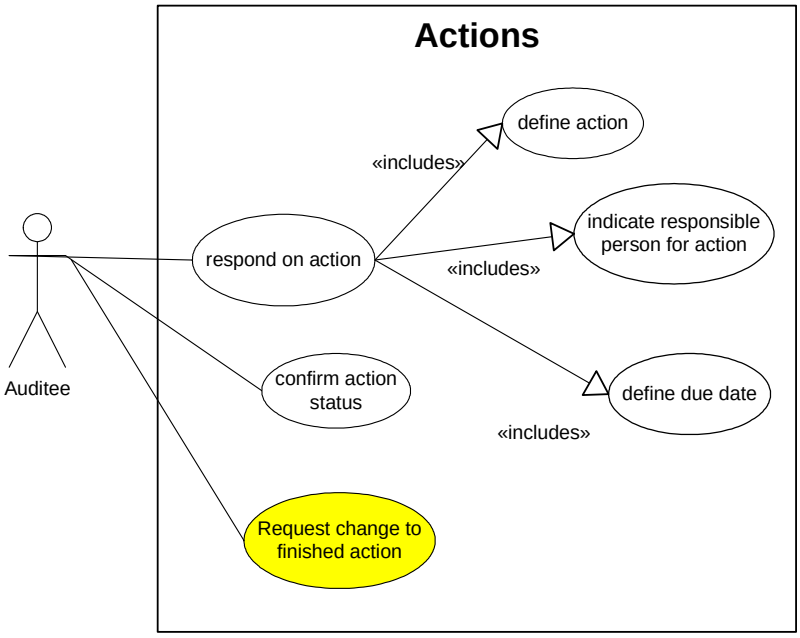


Figure 29 – Package "Actions"

- **Respond on action:** the auditee responds to the deviations with actions to correct it.
- **Confirm action status:** After the execution of the action by the action owner, the auditee has to update the state of the action to identify that it is done.
- **Request change to finished action:** this use case is an improvement suggestion which consists on allowing the auditee to request the authorization to change some detail of an action already closed. This request is sent to the auditor.

4.1.2.14 Package “Administration Operations”

This package contains all the operations that can only be performed by an administrator of the application which are, in the most of cases, CRUD[13] operations. The use cases with yellow background represent those which have to be validated by the competent stakeholders.

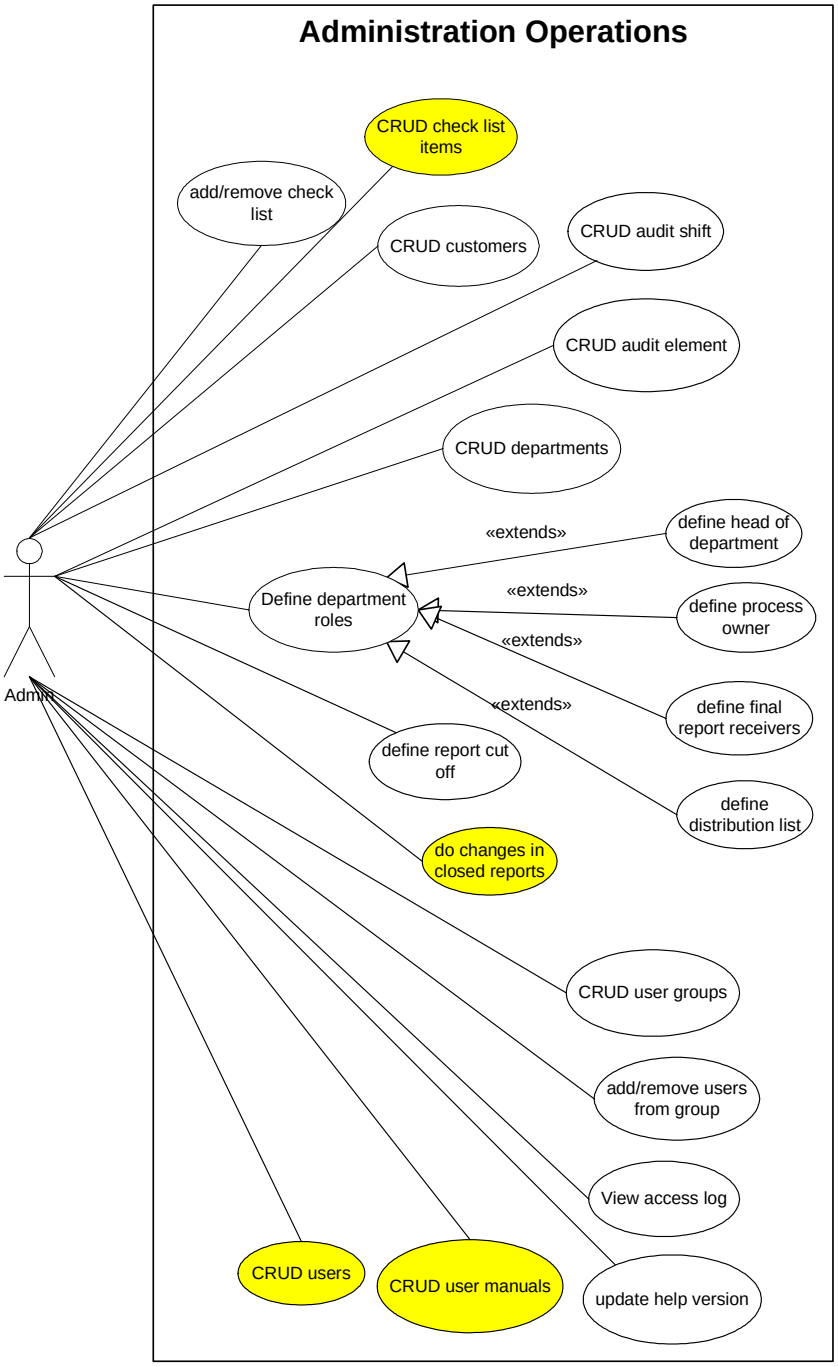


Figure 30 - Package "Administration Operations"

[13] Create, Retrieve, Update and Delete operations for the data stored in the database.

4.1.3 External Interfaces

4.1.3.1 Graphical User Interfaces

The user interface of the new Audit Management application should be similar to the interface of the old application, thus, the users don't have to adapt to a new interface.

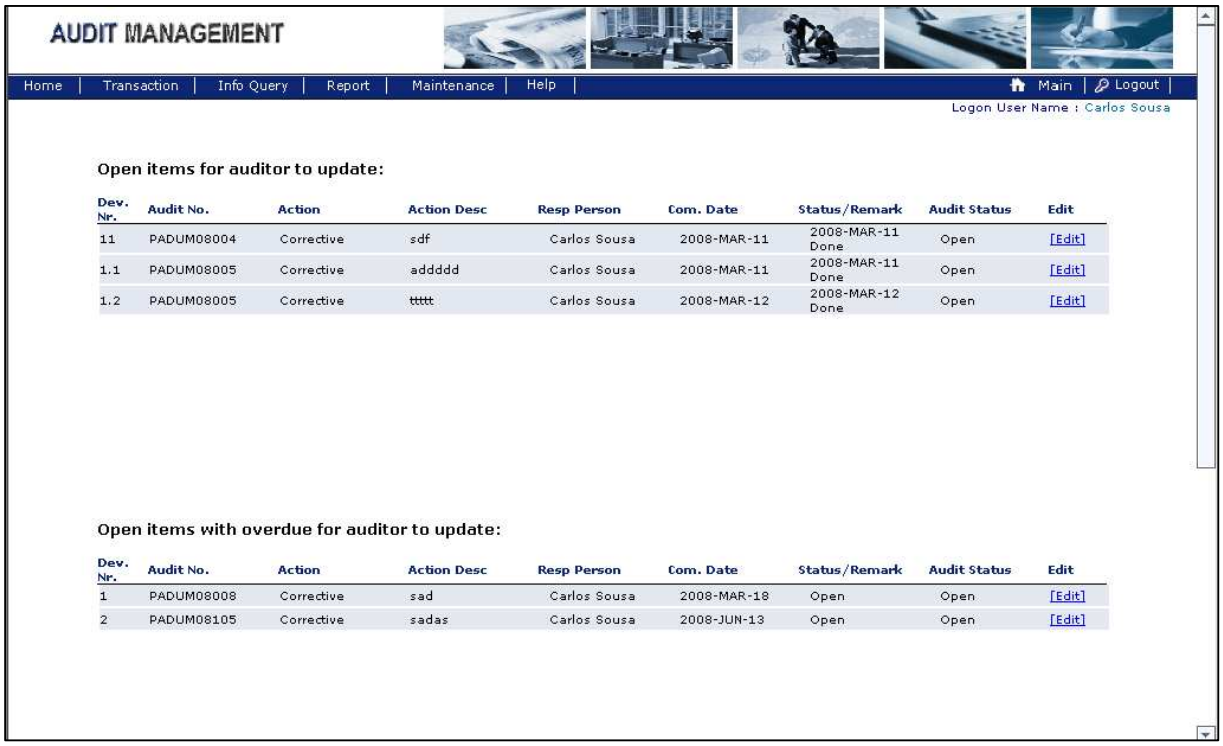


Figure 31 – Layout of Audit Management

4.1.3.2 System Interfaces

The requirements of the project don't include any mandatory interface with another system. However, there are some data needed by the application which is contained in other systems but the way how Audit Management will interact with these external systems is described in the definition of application architecture in the section 4.2 of this thesis.

The users with permissions to access the application must have a valid account in the company network. The data of the users' accounts are stored in an Active Directory[14] system, so, it is necessary to assure that the user who wants to log in the applications has a valid account in this system.

The second particularity of the application that may involve an interface with an external system is the generation of reports. There are several mechanisms that allow the extraction of reports from data fields, so, they may be a possibility to interact with this application.

[14] Technology created by Microsoft that provides a variety of network services including users' authentication.

The company owns some software systems that may be used in the design of the application to reduce the time of the implementation of some features. The usage of these systems is analyzed in section relative to the architecture analysis.

4.1.4 Quality Requirements

All the applications implemented in the company have to be in accordance with high quality patterns. The properties that the application has to meet to be in accordance with the quality requirements are:

- **Reliability:** time needed by application to recover of some crash.
- **Robustness:** system security and low probability of failure.
- **Completeness of function support:** Implementation of all the functionalities specified.
- **Correctness of function support:** Implementation of the functionalities in accordance to their specification.
- **User friendliness:** the usage of the application has to be easy to learn.
- **Time behavior:** time needed to perform some operations (login, queries, etc…)
- **Maintainability:** easiness of maintaining and upgrading the application that can be achieved with simplicity of interface, module independence and complete documentation.
- **Portability:** adaptability to other software and hardware environments.
- **Reusability:** possibility to reuse the product or product parts in other projects.
- **Flexibility:** easiness to make a little modification or update in some feature of the application.

In the case of this project, the most important quality requirements indicated by the users of the application are the time behavior, the maintainability, the portability and the flexibility. The users' complaints were related with the slowness of the application, the difficulty to add some new features, the difficulty to migrate the application from one server to other and the difficulty to make little changes in the application, especially to adapt the audits workflow. To solve these complaints of the users, it has to be given a particular attention to the quality requirements referred.

4.2 Architecture Analysis and Definition

The second phase of the specification of the solution for the reengineering of the Audit Management application consists of analyzing and defining the best architecture to meet the requirements of the application.

In the sections above it is only given an overview about the architecture adopted and the justification of this decision but the whole document of architecture according with the company's template is available in appendix 2.

4.2.1 Overview, requirements and Constraints

The main goal of the architecture of the Audit Management application is to support all the functional and non-functional requirements previously specified. The definition of this architecture takes into account the preferences of the users and of the people responsible for the technical support of the application.

The main preferences referred by the application users are:

- Better performance of the application. The old application takes too much time to perform some operations (login in the application for example).
- The development of new features and the update of some existing ones.

The main preferences referred by the technical support team are:

- More flexibility and modularity of the application. The support team wants to make small adaptations to the flow of the application according to the site where it is installed. Besides that, the team wants to do little customizations in the reports generated by the application. Those changes are very difficult in old application and imply a high effort of the development team.
- The application's ease to install. The old application is hard to install because it depends on several systems and needs several configurations to work properly.

The analysis of the best architecture for the application took into consideration the software systems and tools available in the company with the aim of identifying if their usage was profitable in terms of the application's performance and the development process. So, an analysis of those systems and an evaluation of the benefits and disadvantages of their inclusion in the application architecture were made.

4.2.2 Logical View of Application Architecture

The architecture is logically divided in three main layers according to the pattern Model-View-Controller. Each architecture module, visible in the figure below, groups similar functionalities and provides them to the other modules.

The business layer of the architecture is itself divided in three secondary layers. The top layer contains the code-behind of the web pages, the intermediary layer contains support modules for the other components of the application's architecture and the bottom layer contains components for the data access.

A diagram of the logical view of the architecture is present in the figure below and, after the figure, there's a description about all the components of the architecture.

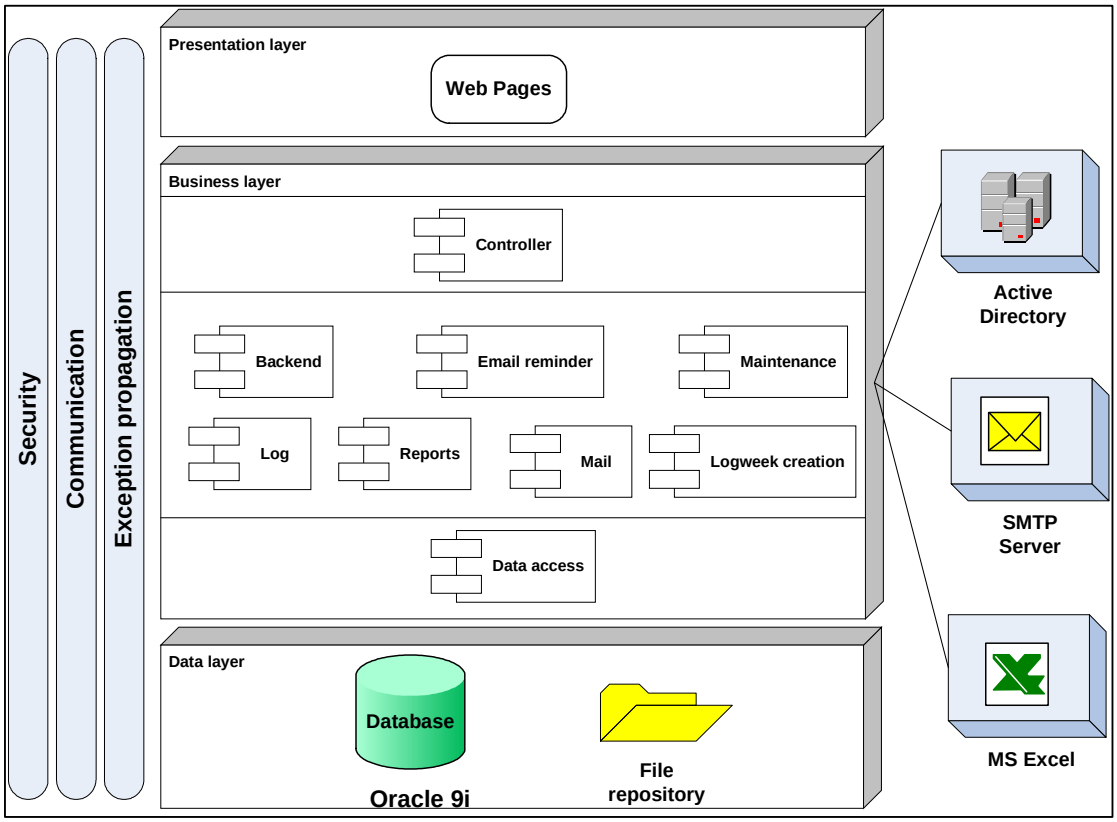


Figure 32 - Logical View of Application's Architecture

- **Web pages:** the user interface of the application consists of a web interface, so, it is accessible across a web browser. The web pages are developed in the .NET 2.0 framework using ASP .NET, HTML, CSS, and JavaScript languages.
- **Controller:** component that contains the code behind the web pages and manages all the business logic of the application, that is, it acts as a coordinator of all the modules of the architecture. It is developed in C# language.
- **Reports:** component that contains functions to manage the generation of reports including their creation, preview, update and deletion. It is developed in C# language and interacts with Microsoft Excel because the reports are generated into Excel files.
- **Maintenance:** component responsible for the management and configuration of maintenance operations. These operations include administration tasks and backend tasks like the deletion of temporary files, the conversion of some pending reports to final reports and the update of the reports data sheet. In the old Audit Management application these operations were executed in a separated executable file, so, when user wanted to perform the maintenance operations he had to run this file. In the new application, these operations are available through the web interface.
- **Backend:** this component is external of the application's core because it is an assembly file (.dll) responsible for performing the backend operations. These

operations may be executed occasionally through the Audit Management application or they may be executed periodically with the support of the Windows Task Scheduler.

- **Backend caller:** it is an executable file that will work as an intermediary between Windows Task Scheduler and Backend.dll, that is, the Windows Task Scheduler will be programmed to run this file periodically and this file will invoke functions present in the Backend.dll file. This file is needed because the Windows Task Scheduler only works with executable files, so, it can not invoke the Backend.dll.
- **Log:** component that contains all the functions necessary to log the main operations done in the application.
- **Mail:** component responsible for the sending of emails and the definition of their contents. In this application there are several emails that have to be sent with different contents and to different recipients according to the type of the email. The generation and the sending of those emails are made inside this component.
- **Email Reminder:** In the logical view of the architecture, this component belongs to the business logic layer but it will be installed in database. It is an Oracle database job responsible to send emails periodically depending on some conditions present in database tables. For example, an email should be sent daily to an auditee if the deadline of the execution of its actions is over. The decision to implement this component in the database is due to questions of performance because all the data necessary to decide if an email has to be sent or not is present in database, so, this functionality is entirely executed in the database. This component is developed with the PL/SQL language in an Oracle 9i database.
- **Logweek Creation:** As the Email Remainder component, it is also physically present in the database because it consists of a database job that has to run yearly. This job fills a database table that contains all the weeks of the fiscal calendar of Qimonda. It is developed in PL/SQL language.
- **Data Access:** component that contains all the functions responsible for all the accesses to the database.
- **Database:** component responsible for the storage of the persistent data of the Audit Management application.

4.2.3 Possible Solutions for Deployment of Application Architecture

The analysis of the architecture of a software application consists of finding the best structure of the components and relationships between them that better meet the previously specified requirements. After the study of the software systems and the software components existing in the company that may be used in the development of the Audit Management application, there are two possible architectures to be adopted.

The deployment of the architecture consists of distributing the logical components into hardware systems. Therefore, in this section the possible deployment views of the architecture of the application are detailed.

Both solutions are able to implement the logical architecture detailed in the previous section and both have their benefits and inconveniences, so, a complete study to determine the solution which has to be adopted was made.

4.2.3.1 Architecture based in YODA

The YODA middleware – detailed in section 4.2.4 – is a powerful software system for distributed applications and provides several features that can be useful for the development of Audit Management. So, the inclusion of this system in the architecture of the application was analyzed. The architecture defined with this system is perfectly feasible and allows the implementation of all the functionalities required for the application.

In the figure below the physical view of the architecture of the application using the YODA middleware is shown. In this image, it is possible to identify all the hardware systems required to develop the application with this architecture.

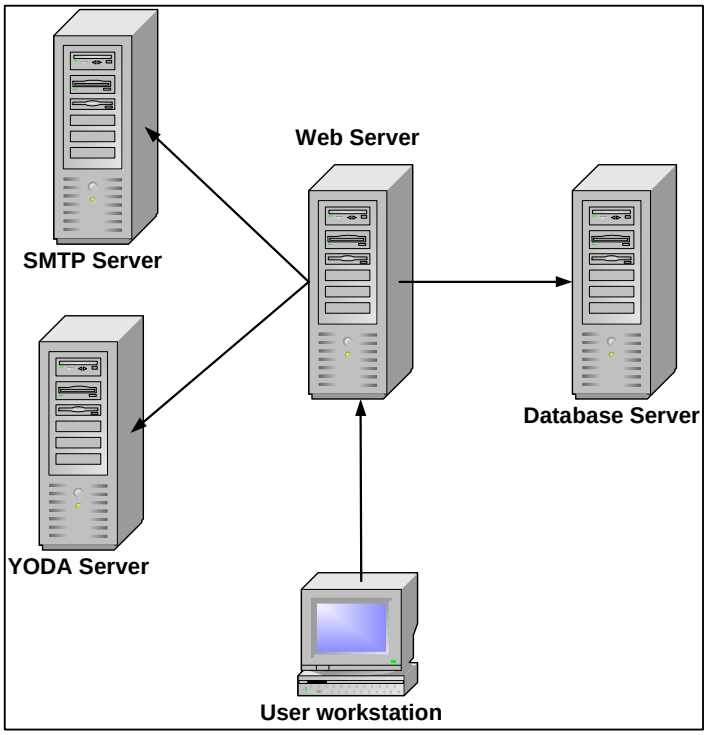


Figure 33 - physical view of architecture based in YODA

The deployment diagram shown below represents a merge between the physical view and the logical view of the architecture. So, in this diagram it is possible to see where the components of the application are physically present.

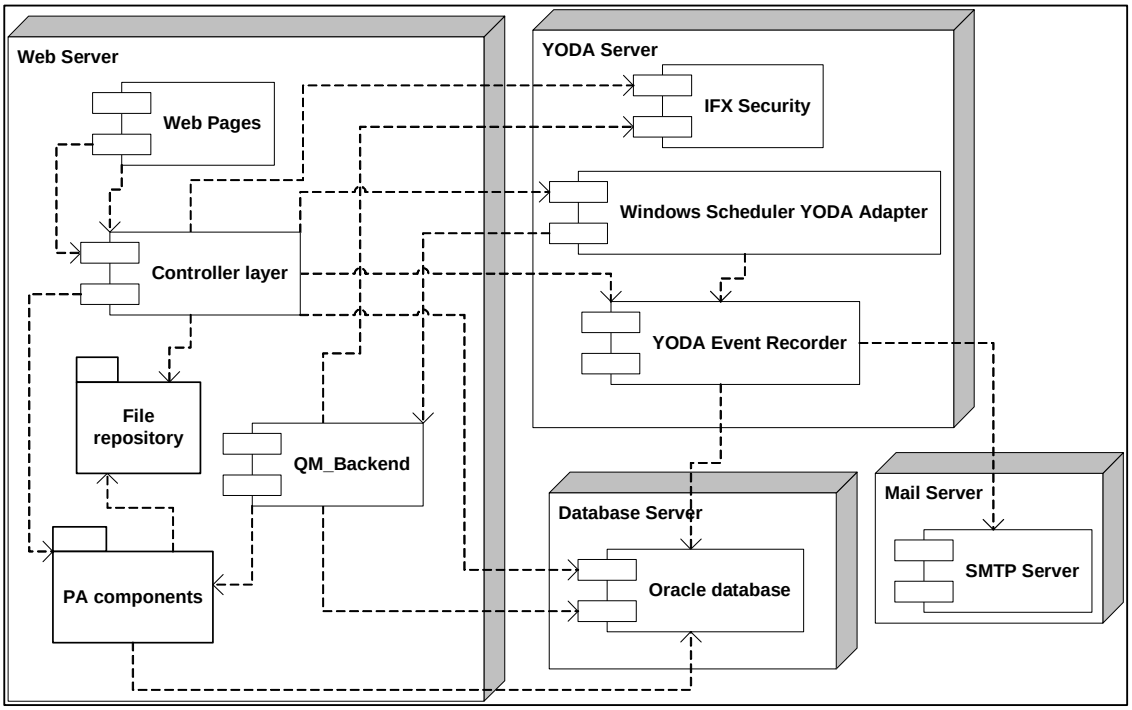


Figure 34 - Deployment view of architecture based in YODA

All the components that are present in the logical diagram in section 4.2.2 are not shown in this diagram because the YODA middleware provides some applications that can replace the absent components. So, the components shown in this diagram that were not shown in the logical diagram in section 4.2.2 are:

- **PA components:** set of COM+ components responsible for the creation and the update of reports.
- **File Repository:** this is not a software component but it is important in the context of the application because the templates for reports, the attachments uploaded by users and the reports generated have to be stored in a file repository.
- **IFX Security:** it is one of several applications available in the YODA middleware. It stores the information of the users and allows the management of users and roles. It also provides functionality for users' authentication.
- **Windows Scheduler YODA Adapter:** this application interacts with the Windows Task Scheduler and provides several high-level functions to configure tasks periodically. It is useful to program the email reminder and the execution of the backend operations.
- **YODA Event Recorder:** application that allows data recording in 4 different systems: database, email, file and Windows Event Log. In the Audit Management application, it is useful for sending emails and for logging operations executed in the application.

4.2.3.2 Architecture for standalone application

A standalone application is an application which doesn't depend on any external system to work. The goal of this architecture is to implement an application as independent as possible from other software systems.

The physical view of this suggested architecture is shown below.

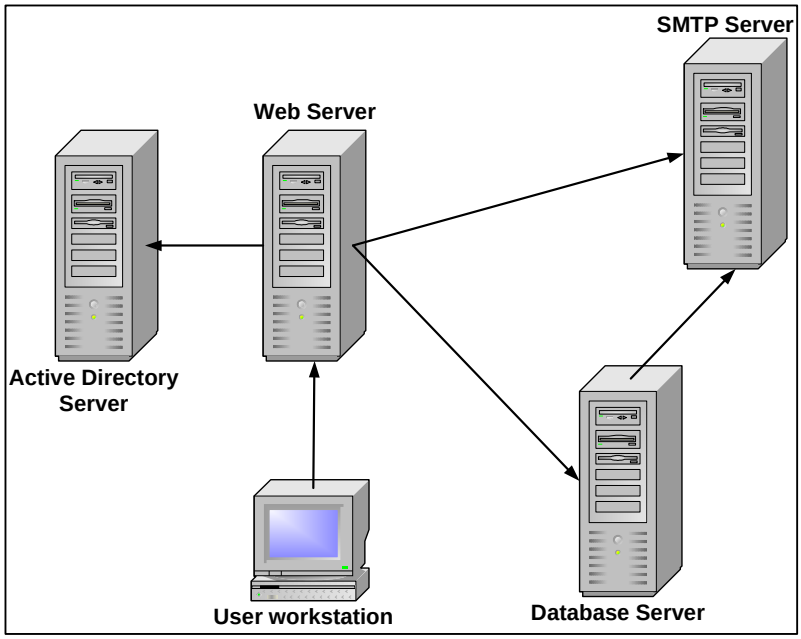


Figure 35 - physical view of standalone architecture

The deployment diagram of this architecture is shown in figure below.

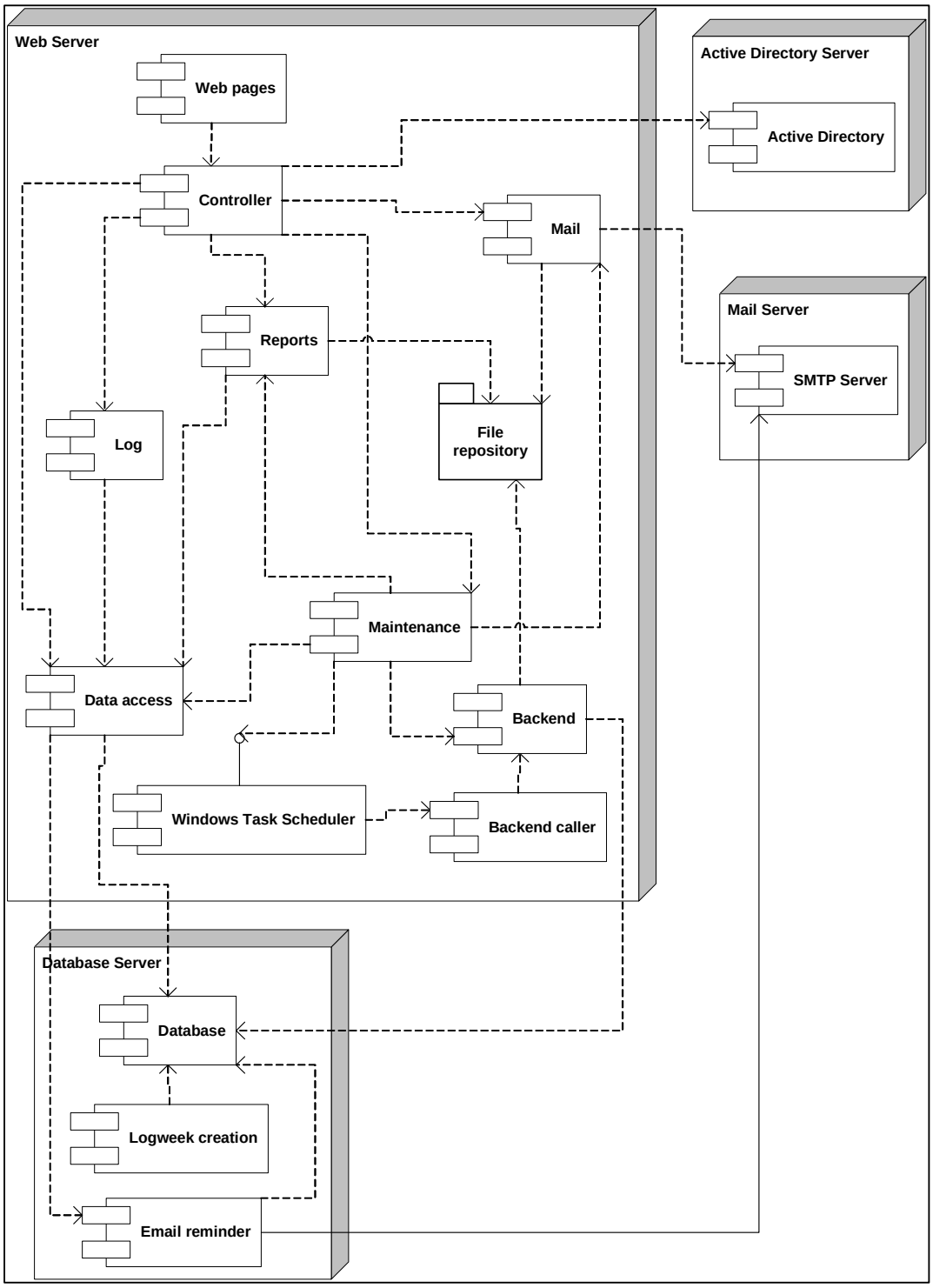


Figure 36 - deployment view of standalone architecture

All the components of the diagram above are described in section 4.2.2 but it is important to refer the existence of the Active Directory component which is used for the authentication of the users.

4.2.3.3 Comparison between possible architectures

The two options for the application architecture presented above have several differences, so, the decision about one of them has to take into account the most important requirements for the company. The analysis of the architecture of Audit Management followed the recommendations of the standard ISO 9126 for software quality: functionality, reliability, usability, efficiency, maintainability and portability.

According to the ISO standard and to the company requirements, the second solution – architecture for a standalone application – was selected.

The main factors that made the decision pending to this solution are:

- **Reliability** – application will depend on less external systems, so, the probability of failures is reduced.
- **Efficiency** – this factor concerns the time behavior and the resources behavior and, particularly in time behavior, this option is better due to few interactions with other systems. The YODA architecture is base in a mechanism of exchange of messages among several applications, so, it is needed more time to perform some operations.
- **Maintainability** – the development of a new feature or the correction of some detail in the application is easier in the standalone architecture than in the architecture based in YODA. Besides that, any upgrade of the Audit Management will be dependent on the compatibility of the YODA system.
- **Portability** – the standalone architecture makes easier the installation of the application because it has fewer components to install and fewer details to configure.

The new application architecture has to support, at least, all the features previously provided by Audit Management with the same or better performance. Besides that, the stakeholders required modularity and flexibility for the source code of the application and these requirements are best achieved with a standalone architecture.

The last aspects that influence the decision about the architecture are the time needed for development and the reuse of source code. The usage of the YODA system would imply the production of more source code to implement some features, so, the development time would be increased.

4.2.4 Technologies

Some of the technologies used in the implementation of the application were imposed by the company due to its policies and its software licenses and others were selected in the phase of the definition of the architecture.

Thus, the main technologies related to the project development are:

Windows Server 2003

The operating system of the server where will be installed the application.

Windows 2000

The operating system of the client computers that will access the application across the web browser.

Framework .NET 2.0

Framework used for the majority of developments within the company and imposed for the execution of this project.

C#

Language of the framework .NET used in the development of the application. It was suggested by the company but not imposed. However, after a little analysis, this language was considered as recommended to achieve the project requirements.

HTML, JavaScript, CSS

Languages used for the development of the web interface of the application.

PL/SQL

Language of the Oracle database in which were implemented the procedures and functions in the database.

COM+

Some COM+ components are used for the management of the reports in the application. They are responsible for the generation, the update and the deletion of Excel files that contain the reports.

4.2.5 Main Design decisions

In the sections below are described some of the design decisions that were taken during the definition of the architecture.

4.2.5.1 Pattern Model-View-Controller

This pattern is constantly recommended as a best practice in the design of software and, in this case, it is indispensable to give to the application the flexibility and extensibility required by the stakeholders. The pattern consists of separating the data needed by application, the business logic code and the user interface code making, thus, code more understandable and reusable.

In the .NET architecture used in this project, this pattern is supported with the following correspondence:

- View – it is implemented in the files with .aspx extension and contains the source code for the web component in the languages ASP .NET, HTML, CSS and JavaScript.
- Controller – contains the code-behind of the web pages written in C# language and manages the events supported by the .NET framework.

- Model – the framework provides mechanisms for the database connections and encapsulates the data within objects that can be used by the business logic code.

4.2.5.2 Configuration

The Audit Management application has a lot of details that may be configurable. The configuration values are maintained in a configuration file (web.config) that confers legibility and easiness to make changes. The values that have to be configured are the database connection, the SMTP connection, the Active Directory connection and the periodicity of execution of the backend operations.

In the database, the jobs responsible for the email reminder and the insertion of the logweeks have a configurable periodicity but this one is configured within the Oracle database.

4.2.5.3 Data Access

The Microsoft .NET Framework provides a class library, ADO .NET, that can be used to access data and data services. In this project, the ADO .NET functions will be used to establish connection with Oracle 9i database and manage the data stored in it.

4.2.5.4 Authentication

Currently, authentication in Audit Management is performed in two steps. First, it is made an YODA authentication, that is, the user only can log on Audit Management if he is present in the IFX Security application. After that, the application verifies if the user is present in the Audit Management database and checks user credentials. This mechanism implies importing users from Active Directory database to YODA and, then, to the application database.

After the application reengineering, authentication will be performed in a different way. To simplify this process and to remove the YODA dependence, the users will be directly imported from Active Directory to the application database. Nevertheless, authentication only in the application database is not enough because the users can be removed from the Active Directory database and still be present in the application database. So, it is necessary to confirm if the user is present in both databases. This problem is solved with the implementation of an auto-login feature. This feature makes the user authentication automatic and transparent for the user and avoids him to enter the username and password already inserted when the user made his authentication in the company network.

4.2.5.5 Logweek creation

The fiscal calendar of Qimonda is different than the regular calendar and the Audit Management application has to interact with the company fiscal calendar. There is a table in the database that makes a correspondence between these calendars and contains all the weeks of the fiscal calendar.

To fill this database table, a database job will be implemented and it will run once a year without intervention of users. This job will update the details of all the weeks for one year.

4.2.5.6 Email Reminder

Currently, this mechanism involves the Windows Task Scheduler to schedule the sending of emails and a COM+ component to send the email. To improve the performance and the application independence, the email reminder mechanism will be implemented in a database job. This job takes the data needed in the database, it compose the email and send it every day until reach the conditions to stop.

Chapter 5

5 Prototype implementation

The implementation of the prototype of the application aims to validate the architecture definition by the implementation of the functionalities specified.

5.1 Implementation Roadmap

As this project is a reengineering project, it has a mandatory requirement which consists of maintaining all the features of the application, that is, the features provided by Audit Management before the reengineering must also be provided after the project reengineering.

The reengineering of the Audit Management application is a complex process due to the application's dimension and to the wide distribution of its components (COM+ components, YODA middleware, Backend operations, email reminder, etc…). To successfully achieve the reengineering project with the maintenance of the functionalities, it is indispensable to do it carefully and progressively and to attempt to always have a functional application.

A roadmap of the implementation was elaborated to anticipate possible problems and to reduce the risks of the development. This roadmap contains the sequence of steps that have to be followed to make a gradual implementation of the application avoiding, thus, the possibility of obtaining a non-functional application.

5.1.1 Overview

The suggestion of the roadmap defined consists of dividing the development process in some phases. Therefore, the approach recommended to the development consists of starting the implementation in the highest logical layer of the architecture – the presentation layer – and ending the implementation in the lowest layer – the database layer.

It is not necessary, and not recommended, to implement all the components at once. So, the best way to maintain the application always functional is to implement some new components and reuse components of the old application. For example, it is possible to implement the web interface of the application and reuse the COM+ components or the email reminder of the old application.

The two following figures show the old Audit Management architecture and the final architecture of the new application. In the old architecture, the components that may be reused in the new application were analyzed before the elaboration of the implementation roadmap with the aim of reusing the majority of the components to quickly obtain a functional application.

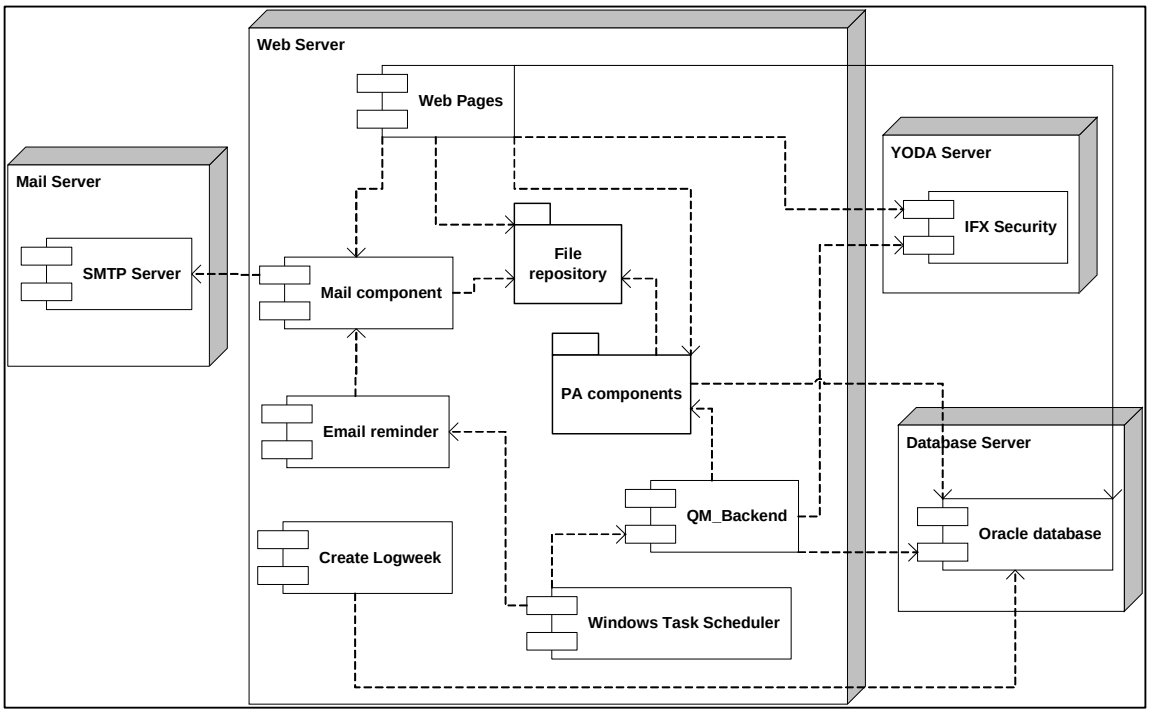


Figure 37 - components of the architecture of the old Audit Management application

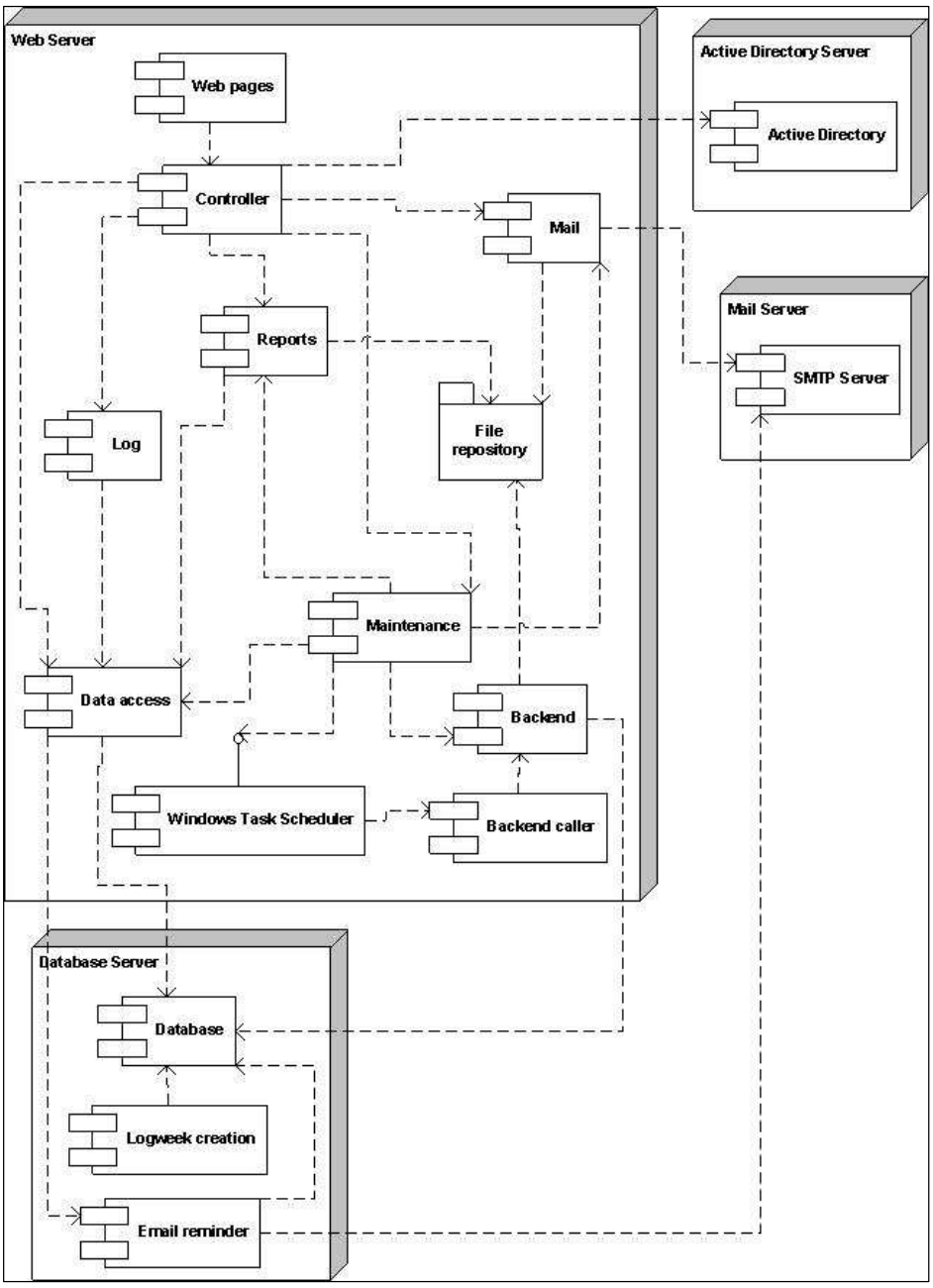


Figure 38 - components of the new Audit Management application

The roadmap of the implementation is divided in four main phases that allow an incremental development of the application. These phases are describes in the following sections.

5.1.2 Core Application implementation

The first step of the roadmap consists of implementing the web interface but the web pages need some components that belong to other logical layers of the architecture. Those components can not be reused from the old application, so, they have to be developed in this phase. Basically, in this phase all the components that can not be reused from the old application are implemented.

The components shown in the deployment diagram in the previous section that are implemented in this phase are:

- Web pages
- Controller
- Mail
- Data access
- Log

The authentication mechanism is also implemented in this phase.

These new implemented components interact with components of the old application, so, at the end of this phase an intermediary deployment diagram of the architecture as shown below is obtained. The components with yellow background are the new implemented components, the components with grey background are those reused from the old application and the components without background haven't to be implemented.

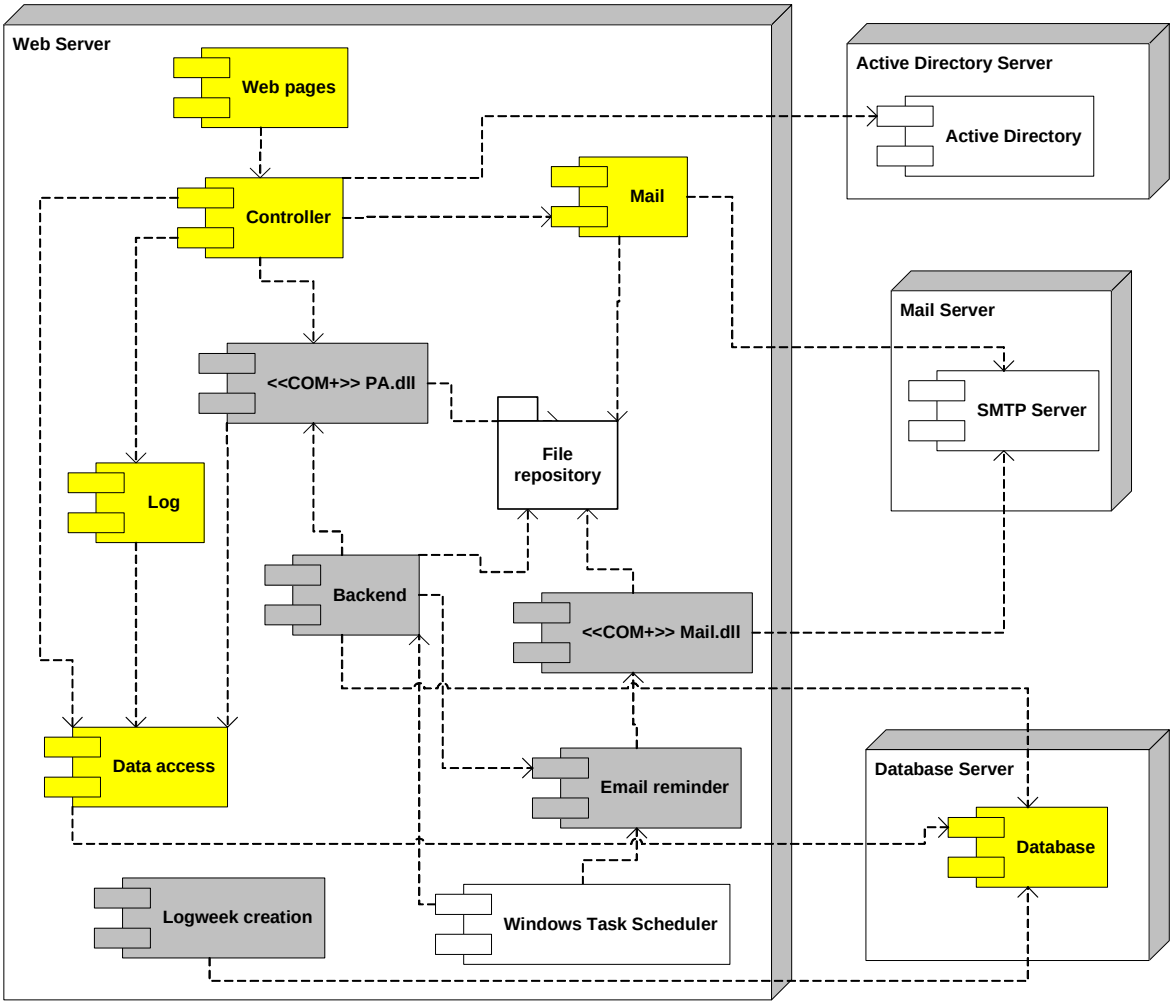


Figure 39 - intermediary deployment view of the architecture

5.1.3 Implementation of external features

The second phase of the implementation concerns the components that don't belong to the application core but provide functionalities for support and maintenance. Some of the components are developed to replace the components of the old application and other components are implemented to add new functionalities in the application.

The components that have to be implemented in this phase are:

- Maintenance
- Backend
- Backend caller
- Email reminder
- Logweek creation

The description of these components is available in the section 4.2.2 of this thesis.

5.1.4 Reports Generation Analysis and Migration

After the two previous phases of the implementation, only the database and the COM+ components are not replaced. The COM+ components are responsible for the generation of reports, so, the third phase of the implementation consists of analyzing the best way to generate the reports.

The best solution to make the reports generation more flexible and with a better performance was the object of a complete study which is detailed in chapter 6. In this study, several solutions for reports generation were analyzed including keeping the old COM+ components. If the study had determined that the best way to implement the reports generation implied any development, it would have been developed in this phase.

5.1.5 Database Analysis and Upgrade

The last phase of the implementation of the application consists of analyzing a possible restructuring of the database. In this phase, the benefits and the inconveniences of the redesign of the relational model of the database were analyzed.

The analysis of the database concluded that it is well designed and adequate to support the layers above it. The development of the other components of the application allowed the testing of the database and its integration in the application architecture and the results were satisfactory. Besides that, the reengineering of the database would imply a strong effort that would not bring enough benefits to justify it. Therefore, the database structure was kept as it was.

5.2 Application implementation

The first goal of the implementation of the application is to be a proof of concept of the application architecture in terms of modularity, flexibility and performance. However, this project was not planned to stop with the implementation of a simple prototype with some functionalities that prove the feasibility of the project. If the prototype confirms that the application may be implemented with this architecture, all the features have to be

implemented and, effectively, after the validation by the prototype the other features were added to it.

5.2.1 Description of the application development

The development of this application had some particularities due to the type of project and its contextualization in the company's requirements.

This application was developed with the IDE Microsoft Visual Studio 2005 in ASP .NET and C# languages. The company has its own rules and guidelines for its projects, so, this project was developed following the coding standards of the company for the C# language. Besides this, IBM Rational ClearCase that is the tool existing in the company for the control of the versions of the source code was also used.

The development had always taken into account the specification of the functional and non-functional requirements and the architecture previously defined. Nevertheless, there are several rules that were embedded in the source code of the old application, so, before the development of a new web page or other feature, the source code correspondent to that feature in the old application was analyzed in detail.

To avoid the development of several features without testing their functionality, a mechanism that provide an immediate first overview of the success of the implementation of the feature was adopted. The functionality developed was tested in both the old and the new Audit Management application and the results of the execution of the functionality and the behavior of both applications were compared. Although this comparison doesn't confirm that the feature works properly, it immediately discards the feature if the results are not identical.

5.2.2 Main implementation details

The old Audit Management application contained several rules embedded in the code but the existence of rules hard-coded is a practice that has to be avoided because it degrades the application's maintainability and represents information which is difficult to access. The same happens with some configurations of the application like the connection strings to the database or the address of the SMTP[15] server, so, in the development of this project, all the configuration values were put in a configuration file.

To improve the modularity and flexibility of the application, the code that may be used several times was encapsulated within a function. The methods that are related with the same features were also grouped into the same class (e.g., the methods responsible for composing, formatting and sending emails are all in the class *Mail*). The same happens for the layout of the web pages of the application because a master page was created and all the web pages are built over it.

The data structure most frequently used in the development of this application is the Dataset present in the .Net 2.0 framework. This structure may be represented as a collection of tables as shown in the figure below.

[15] Simple Mail Transfer Protocol – it is the standard for email transmissions across the internet.

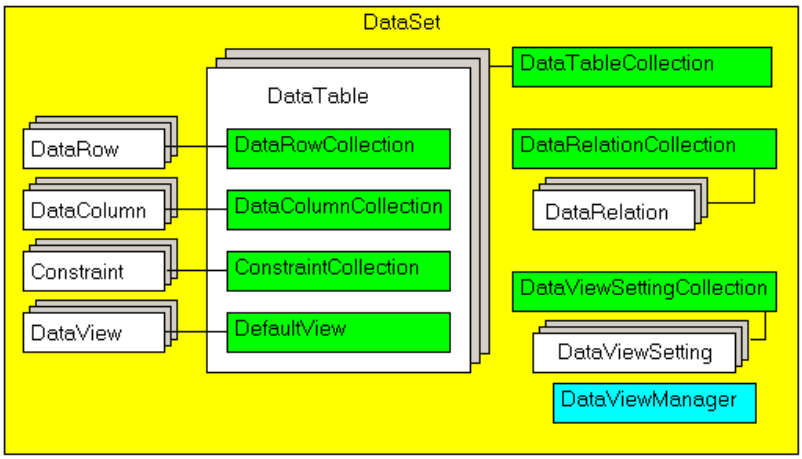


Figure 40 - Dataset structure

The Dataset is used to store the values returned from the database and it is often combined with a GridView from the .NET framework. A GridView is a component that shows in the web interface a tabular view of data, so, the binding between the Dataset and the GridView is frequently used to show data stored in the database.

A specific class to contain all the methods for the access to the database was also developed. Several methods were developed depending on the type of access to the database. There are 4 different methods for the accesses to the database:

- A method that executes a query that returns some values (*Select*)
- A method that executes a query that doesn't return values (*Insert, Update or Delete*).
- A method to execute stored procedures that returns values.
- A method to execute stored procedures that doesn't return values.

The connection to the database is made across the Microsoft provider (MSDAORA) for the API[16] Ole DB. Each access to the database is made within a transaction that will rollback if the operation in the database doesn't succeed.

5.2.3 Implementation Results

The results of the implementation may be divided in two phases. In a first phase, the implementation of a prototype allows validating the architecture defined and proves that the implantation of the functional and non-functional requirements is possible with this architecture. In a second phase, other features were added to the prototype to implement the complete application as specified in the requirements specification document.

[16] Application Programming Interface – set of declarations of functions that may be invoked by the programming languages.

All the implementation was made with Microsoft Visual Studio 2005 where a Web Site solution was developed. The left figure below shows the components of the Visual Studio solution and, in the right figure, the relationships between those components are shown.

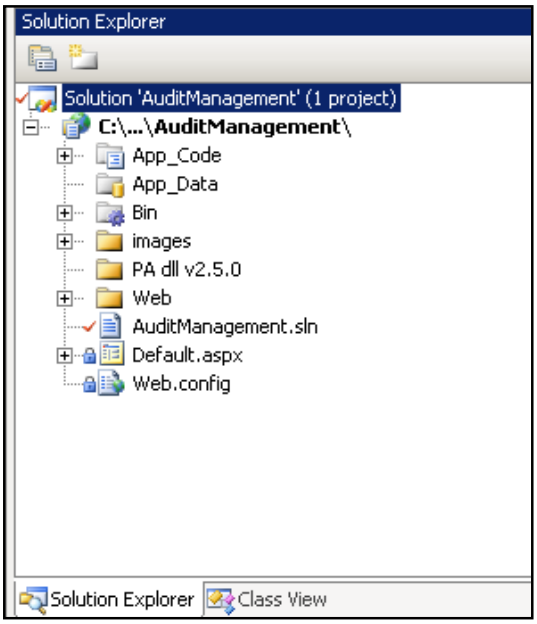


Figure 41 – Visual Studio Solution

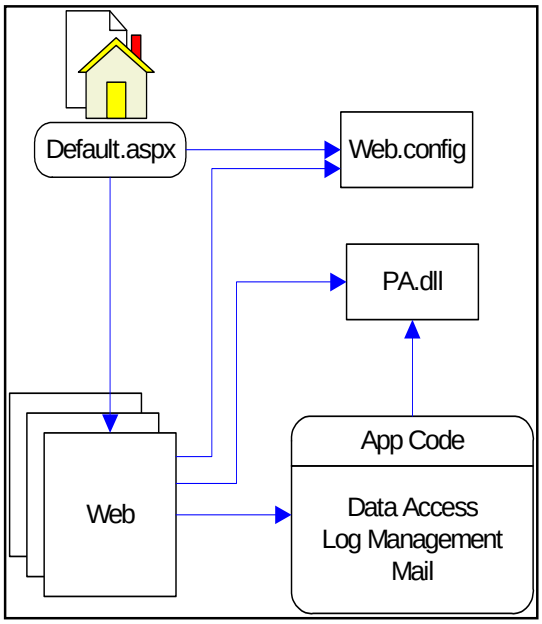


Figure 42- Organization of the Solution

The components of the solution are:

- Default.aspx – first page of the application.
- Web – folder that contains all the source code files for the web pages.
- Web.config – file that contains some configuration values (connection string, address of SMTP server)
- PA.dll – library that contains functions for the management of the reports. It is installed as a COM+ component in the operating system.
- Data Access – file that contains the functions for the accesses to the database.
- Log Management – file that contains the functions to manage the logs of the operations in the application.
- Mail – file that contains the functions responsible for the management of the sending of mails.

The following sequence of figures shows some windows of the application.

This screen corresponds to the page of the creation of a new audit. The user enters some details like the audit category, the audit date or the name of the auditor and submits the information into the system.

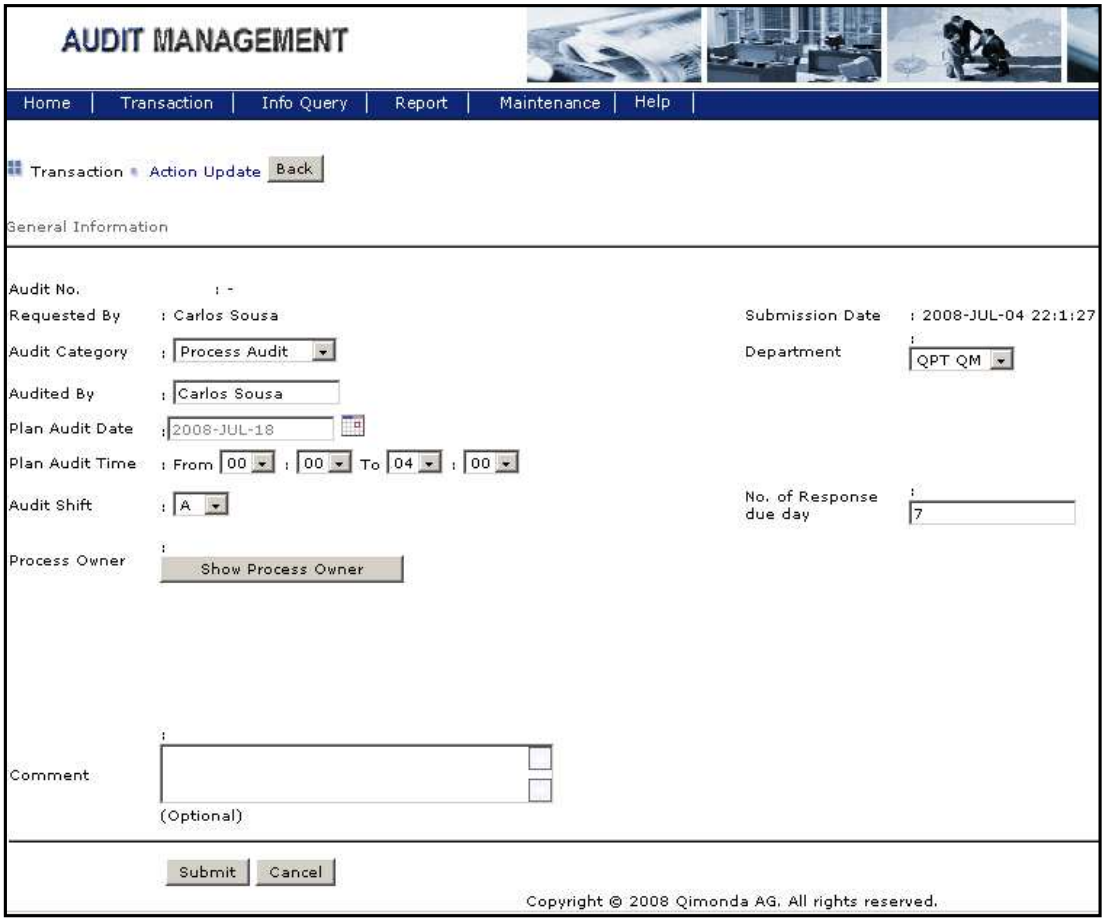


Figure 43 - Creation of new audit

After the request of a new audit by the auditor, the auditee has to accept or reject the audit schedule proposed by the auditor.

Figure 44 - Accept audit schedule

If the auditee rejects the initial audit schedule, he has to propose a new one to the auditor.

Figure 45 - Reject audit schedule

The next step of the audit workflow consists of executing the audit and adding the deviations detected.

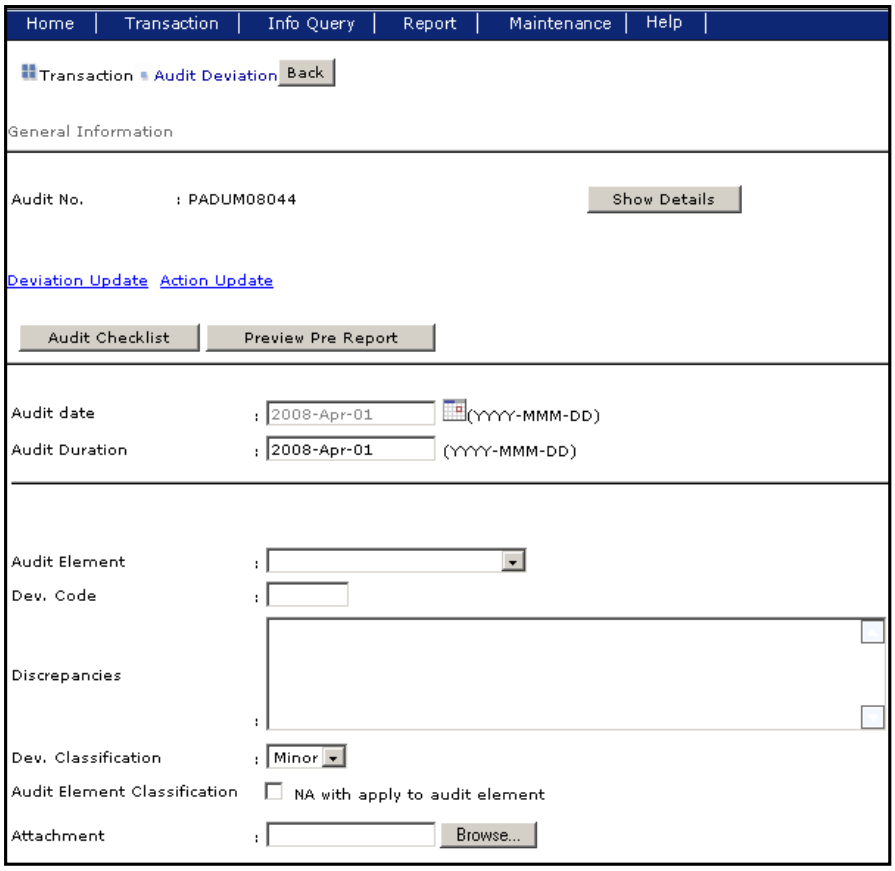


Figure 46 – Add deviations

To correct the deviations, some actions have to be defined and a person to execute the action has to be assigned by the auditee.

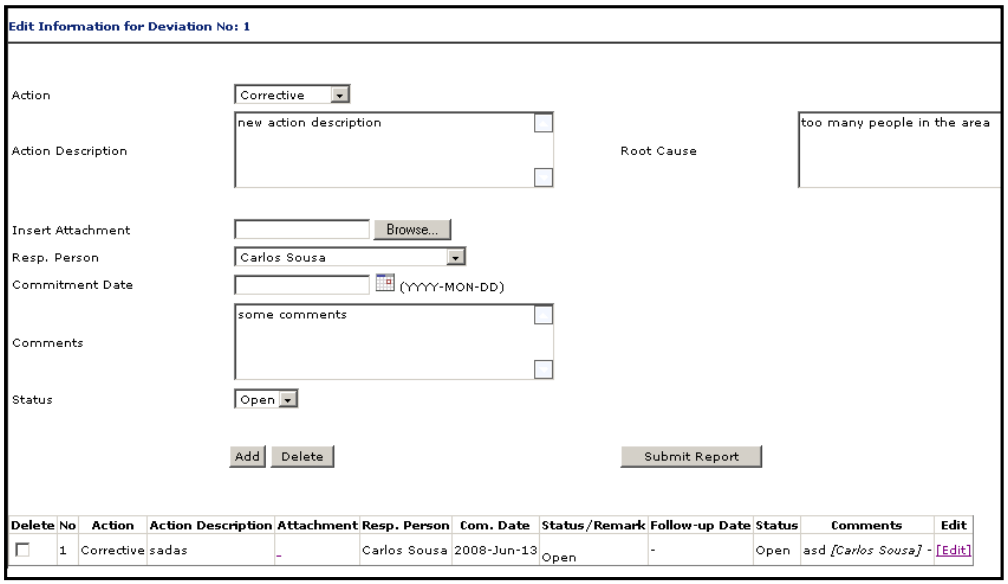


Figure 47 - Add action

The following screen corresponds to the page where the search of audit is made. The user may filter the audits by some criteria available in the page. If no criteria are specified, all the audits existing in the system are displayed.

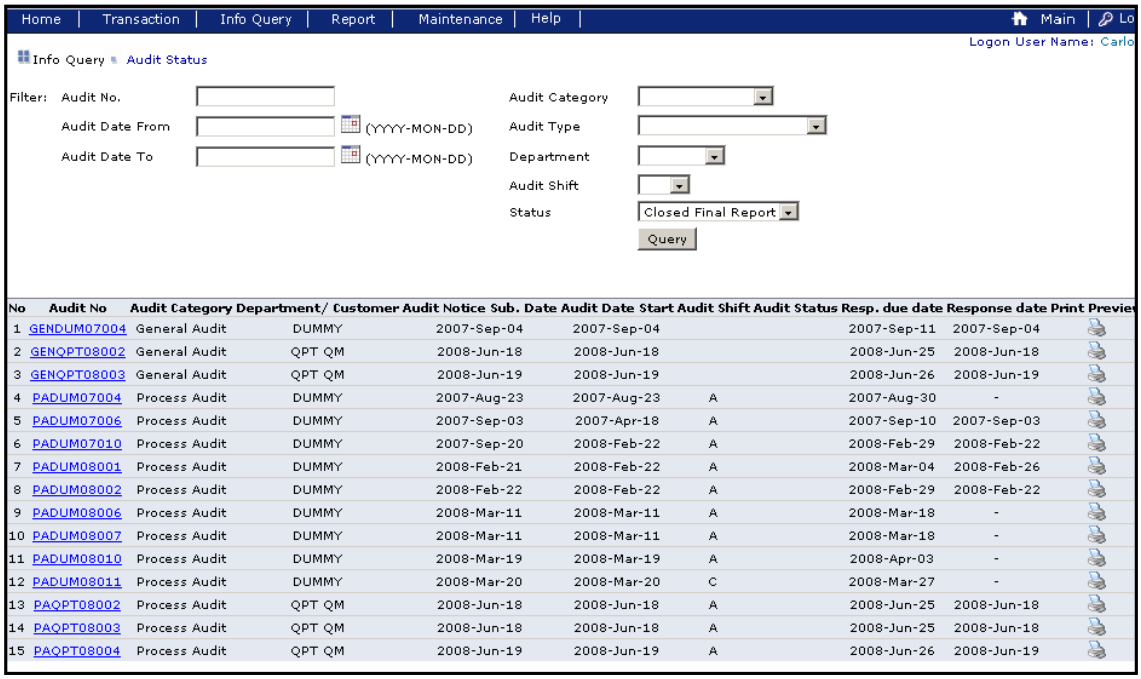


Figure 48 - Audit search

The page where the reports are generated is shown in the next figure.

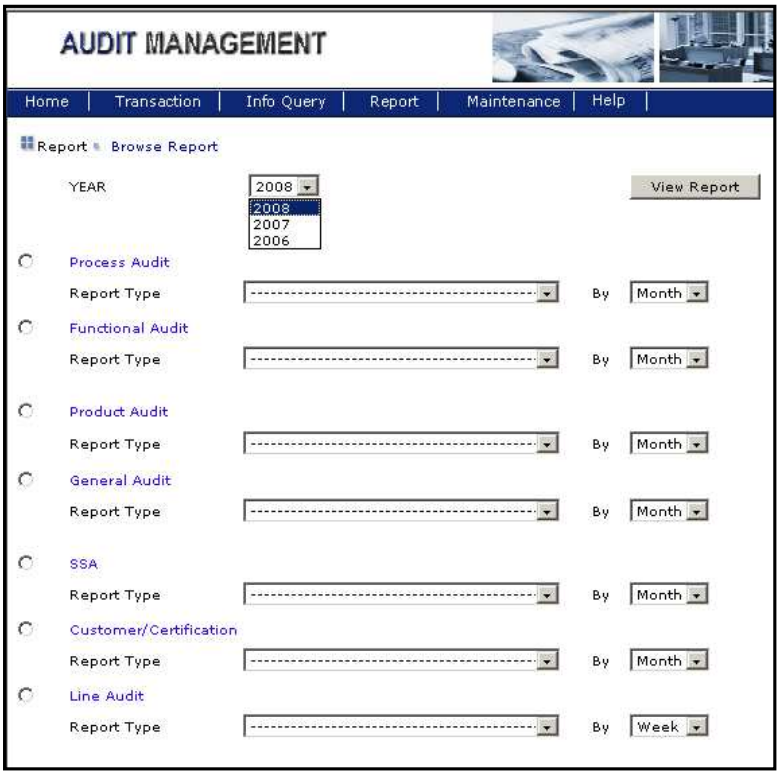


Figure 49 - Browse reports

The following screens show some operations that can only be performed by an administrator of the application.

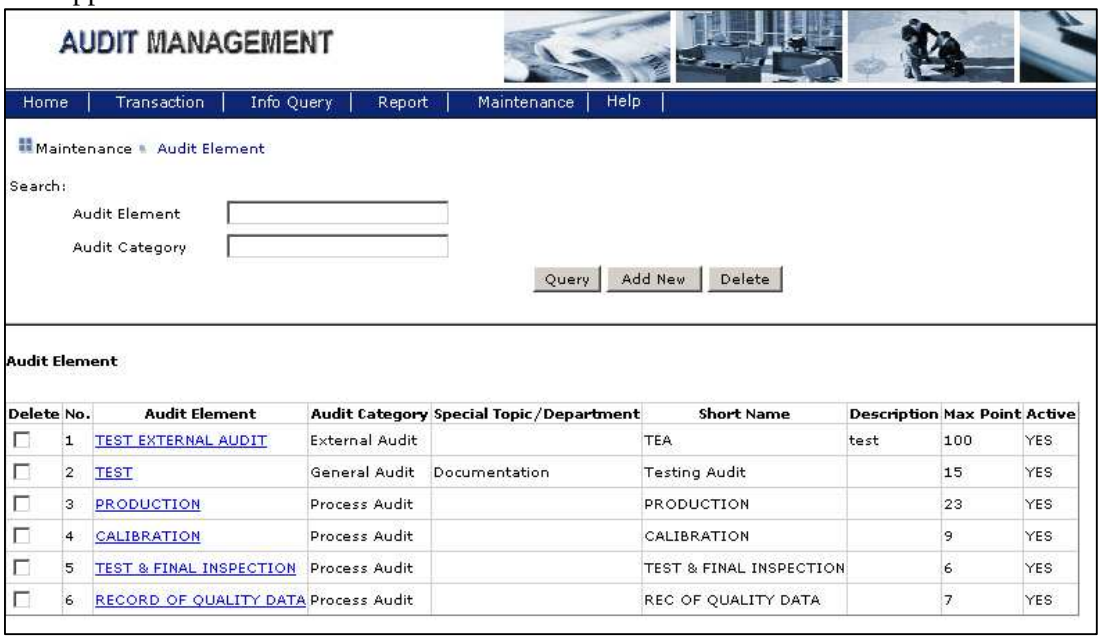


Figure 50 - CRUD audit elements

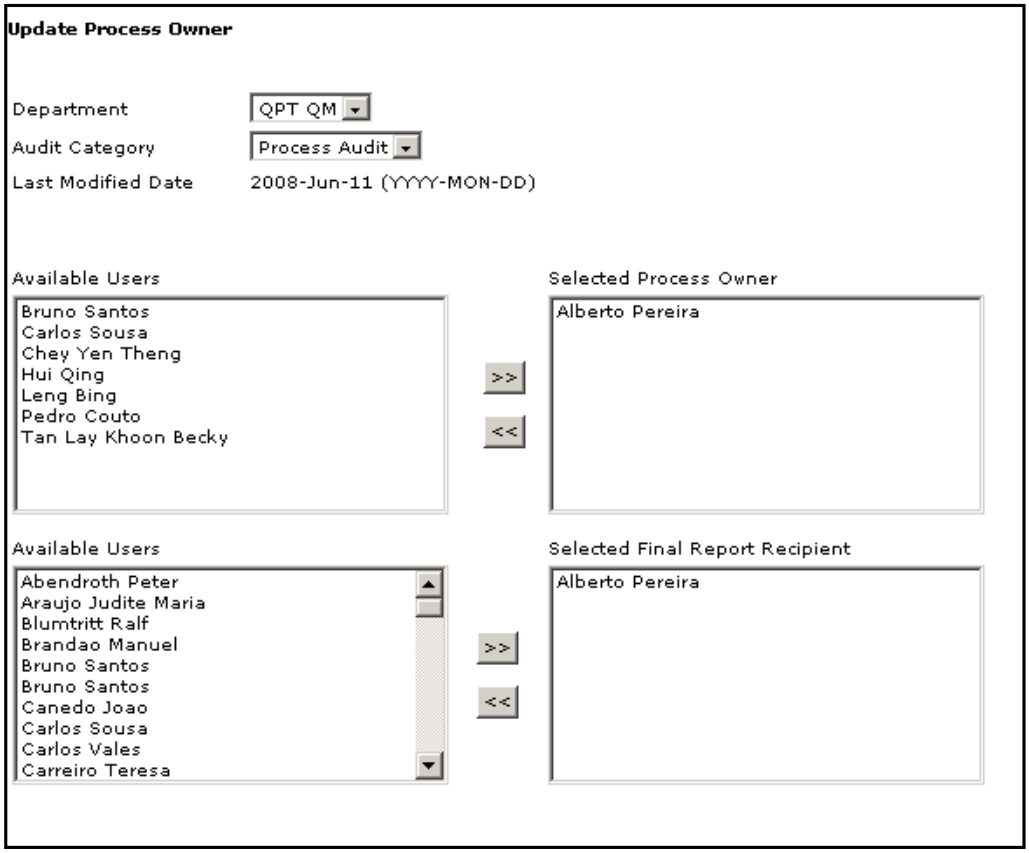


Figure 51 - Update process owner

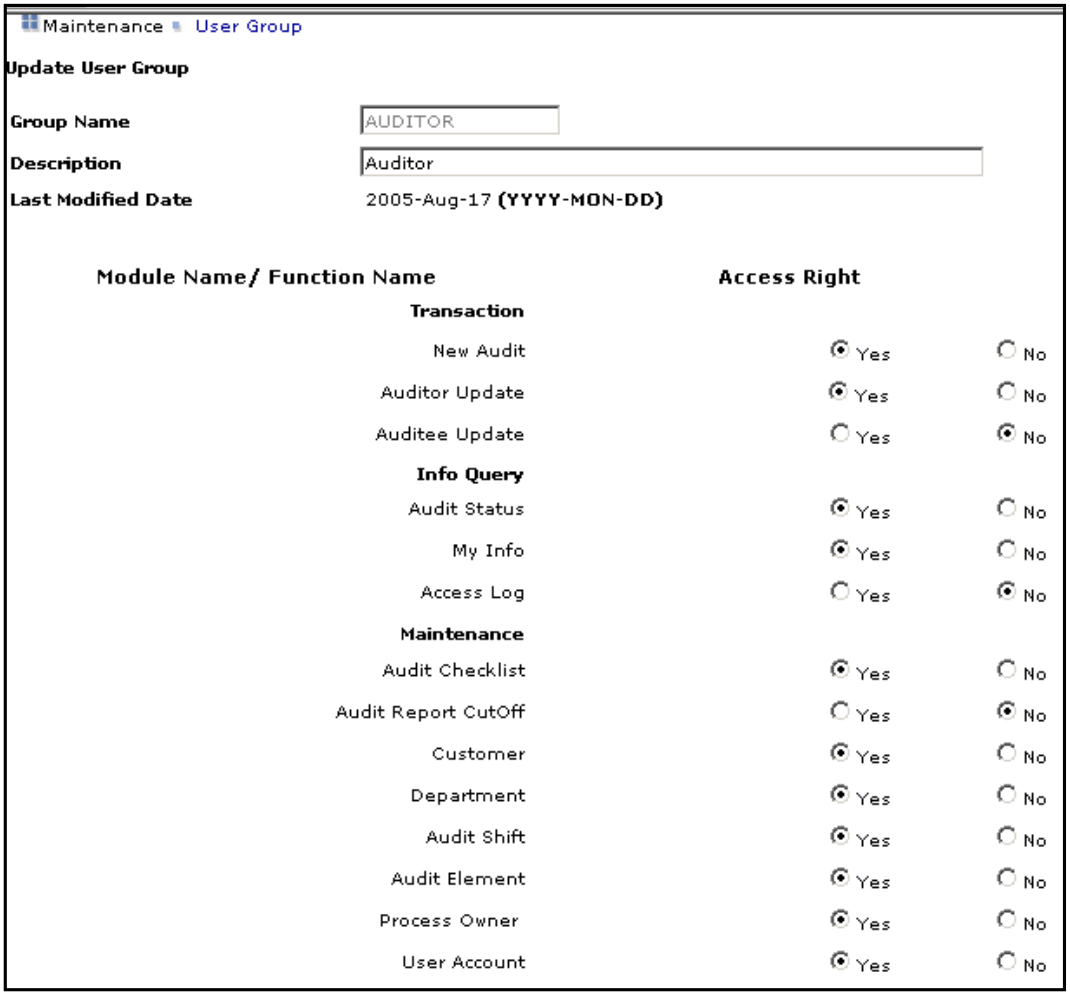


Figure 52 - User group permissions

Chapter 6

6 Analysis of features for application's evolution

After the implementation of the Audit Management application, three analysis to evolve the application were done. The goals of the analysis were to identify the profits and the feasibility of the proposed evolutions or replacements of some features.

6.1 SAP AUD evaluation

6.1.1 Objectives

The purpose of this analysis is to identify the best way to give technological support for the management of Qimonda audits. Currently, Qimonda has its own audit management solution used in some sites by people of quality department. This solution has some benefits and disadvantages which will be described in the next chapters. Since SAP is used in Qimonda and it provides a module for the management of audits, it makes sense to analyze and compare both solutions and decide what shall be adopted.

As it was shown in previous chapter, SAP AUD and Audit Management are the two remaining applications in Qimonda that support the management of audits, thus, with this analysis, it will be possible to decide if Audit Management application could be replaced by SAP AUD without loss of functionality and quality.

6.1.2 Solutions Description

6.1.2.1 Audit Management Application

Audit Management application is Qimonda ownership and was developed for the IT department in Malacca site (Malaysia) around 2003 before the Qimonda spin-off from Infineon Technologies. Currently, this application is installed in Malacca and will be installed soon at Qimonda sites in Porto, Senaï(Malaysia) and Suzhou(China).

The development of this application was already closed and the team responsible for it is no longer allocated to the project. Thus, any request for more developments in the application will trigger the usual steps in these situations, this is, the business owners of the application in all sites will be consulted to approve the request and, after that, the specification document will be written and delivered to a Development Center.

Audit Management is a web-based application available across the Qimonda intranet for users registered in its database. Thus, the users don't need to install anything in their computers except a web browser if it is not installed yet.

6.1.2.2 SAP AUD Solution

SAP Business Suite is a set of software solutions for enterprise processes and businesses. This suite is structured like a tree due to its numerous applications and solutions. Descending in the tree, it will be found SAP AUD module that is the SAP solution for the management of

audits. In the figure below are highlighted the modules or packages that a company have to buy to user SAP Audit Management.

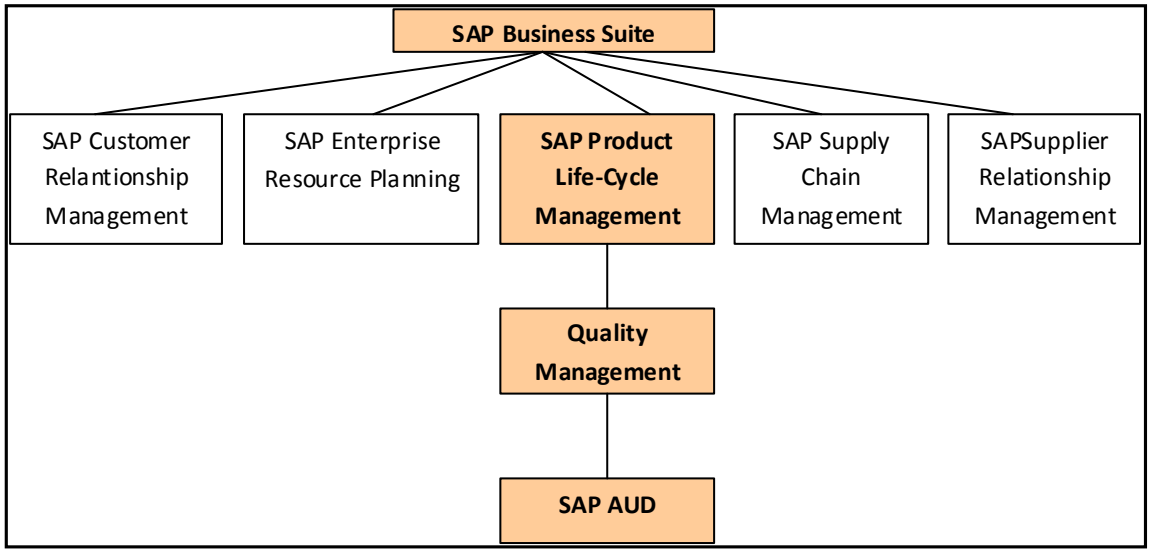


Figure 53 - Organization of SAP suite of solutions

SAP system is used in Qimonda for the management of some enterprise processes and business but, when SAP began to be implemented in the company, there were other standalone applications responsible for these tasks. Progressively, the functionalities of some applications were migrated to the SAP solutions.

In the case of the management of audits, SAP provides the SAP AUD module which covers a large set of audit types but, currently, this solution is only used for the qualification of suppliers by people of quality department in all sites.

SAP AUD, as other SAP solutions, requires the installation of a client program, with a graphical user interface, for its usage. After installation, the authorized users could access the application across a connection to the server where is stored the application data.

6.1.3 Functional Analysis

This analysis was made considering the actual state of applications but it's possible to develop more features or change some details in both. These developments have different costs associated which will be detailed later in this document.

6.1.3.1 Features Listing

At functional level, SAP AUD is more complete than Audit Management and has a very large set of features but, to adopt SAP solution, it is mandatory that all the old functions are maintained. In the table below are presented the most important functionalities for the management of audits.

Functionality	Description	Audit Management	SAP AUD
Audit Creation	Insertion of a new audit in the system	yes	yes
Add attachments to audit	Assign documents to an audit.	no	yes
Audit Schedule Negotiation	Negotiation of audit date and time between auditor and auditee.	yes	no
Add Deviation by Auditor	Add in system deviations found during audit execution.	yes	yes
Update Deviation by Auditor	Update deviation information.	yes	yes
Add attachments to deviations	Assign document to deviation.	yes	yes
Compilation of Preliminary Report	Compile a first version of audit report to be presented to auditee.	yes	no
Preliminary Report Negotiation	Possibility for auditee to disagree with deviations identified by auditor in preliminary report and requests a new report compilation.	yes	no
Add Actions by Auditee	Insertion of actions by auditee to correct some deviation.	yes	no
Update Actions by Auditee	Possibility for auditee to update information about actions.	yes	yes
Add Actions by Auditor	Insertion of actions by auditor to correct some deviation.	no	yes
Update Actions by Auditor	Possibility for auditor to update information about actions.	no	yes
Actions follow-up by Auditor	Verification and approval by auditor of actions performed.	yes	yes
Actions repetition	If action result is not approved by auditor it should be repeated and the system creates a new action with same characteristics.	yes	yes
Close of Final Report by Auditor	Auditor indicates that audit report is closed and no more changes can be done in it.	yes	yes
Email Reminder	Email is sent to the person responsible for some action a few days before action due date and all days after this date if action is not completed.	yes	yes
Audit Search	Search for audits with some filters.	yes	yes

Audit Report	Creation of audit report with relevant information about audit.	yes	yes
Statistical Reports Generation	Generation of some reports with statistics based in audits performed.	yes	no
Report Search	Search for reports with some filters.	yes	yes
Report data update	Update reports content with data of new audits.	yes	yes
Audit Results Valuation	Valuation of audit results based on classification of each item in checklist.	yes	yes
Add/Remove Checklist	Add and remove a new checklist for audits.	yes	yes
Add/Remove Customer	Add and remove customer authorized to perform audits.	yes	yes
Add/Remove Department	Add and remove departments where audits can be performed.	yes	no
Add/Remove Audit Shift	Add and remove shifts when audit can be performed.	yes	no
Add/Remove Audit Element	Add and remove audit elements for evaluation in audits.	yes	yes
Define Process Owner	Define user responsible for department or process where audit is performed.	yes	yes
Create user roles	Create roles for users which can use application.	yes	yes
Define user role	Assign a role to user.	yes	yes
Create Audit Plans	Create a detailed plan for audit execution.	no	yes
Definition of templates	Define templates for all audit objects (audit plan, checklist, etc...) which can be used in several audits.	no	yes

Table 1 - Features comparison between SAP AUD and Audit Management

6.1.3.2 Audit Categories

The table below contains the categories of audits performed in Qimonda. All these categories are supported by Audit Management. In the case of SAP AUD it was considered the standard version of application but it is possible, with a small customization, to add new audit categories and attributes to these categories.

Audit Category	Audit Management	SAP AUD
Process Audit	yes	yes
Functional Audit	yes	yes
General Audit	yes	yes
External Audit	yes	yes
Line Audit	yes	no

Table 2 - Audit categories comparison

6.1.3.3 Report Types

There are two kinds of reports: audit reports and statistical reports. An audit report contains all the information about one audit like the name of the auditor, the department where audit was done, the date and time when audit was executed, the classification of the checklist items, etc… The statistical reports are generated with data of several audit reports and provide an overview about the results of all the audits performed. These reports are, usually, in form of diagrams and graphs.

Report Type	Audit Management	SAP AUD
Audit Valuation	yes	yes
Audit Result by event & score	yes	no
Action Overdue (%)	yes	no
Response Time overdue (%)	yes	no
Response time (Min, Avg, Max)	yes	no
Cycle time (Min, Avg, Max)	yes	no
Improvement	yes	no

Table 3 - Report types comparison

6.1.3.4 Key User roles

An audit involves several persons with different roles and responsibilities. In the next table are present the users required in all audits, independently of the audit category or type.

Key User	Description	Audit Management	SAP AUD
Auditor	Person responsible to perform audit.	yes	yes
Auditee	Person responsible for department or section where audit is performed.	yes	yes
Action Owner	Person assigned to achieve action.	yes	yes

Table 4 - User roles comparison

6.1.3.5 Audit Workflows

Depending on their type, audits have different workflows in the Qimonda environment. Current Audit Management application supports the following workflows and, in this section, it will be identified if SAP AUD also supports these workflows.

- Process Audits and Functional Audits

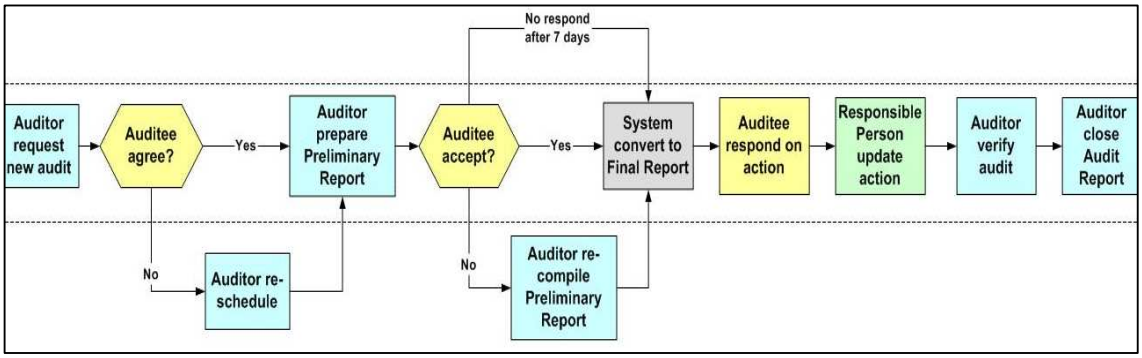


Figure 54 - Process and function audits workflow

This audit flow includes two negotiations between auditor and auditee. The first one is relative to the audit date and time proposed by the auditor and the second one is about the preliminary audit report. In both cases auditee may use Audit Management application to ask auditor to review its initial proposals.

SAP AUD solution does not allow these possibilities, so, if the auditee has some remark to do to auditor decisions, he has to do them *offline*, this is, without any record in the application.

All the others steps of this flow may be supported by SAP AUD.

- General Audits and Functional Audits

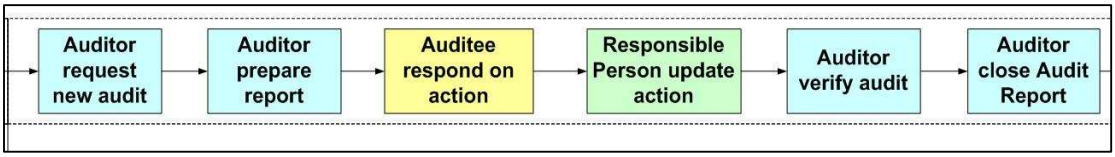


Figure 55 - General and external audits workflow

The general audits and the external audits have the same flow and don't allow discussion about the audit schedule or the deviations between auditor and auditee. Thus, this flow is supported by Audit Management and by SAP AUD.

- Line Audits

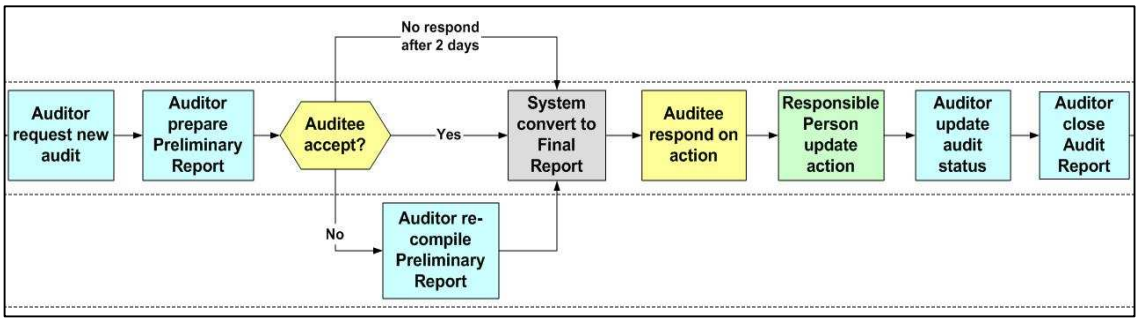


Figure 56 – Line audits workflow

The flow of this audit type includes the possibility for the auditee to do some remarks about the deviations present in the preliminary audit report and request auditor to review these deviations. In SAP AUD solution, this step will not be recorded in the application and should be done *offline*. The other steps are available in SAP AUD.

6.1.4 License and Acquisition costs

Audit Management application belongs to Qimonda since several years and, consequently, it doesn't have any cost for licenses or purchase.

SAP solutions have several types of licensing packages with different prices but Qimonda purchases a complete SAP system with all the software solutions provided by SAP and includes, evidently, SAP AUD application. Thus, the usage of SAP AUD will not bring more costs to Qimonda.

This topic will not be influent in the decision about one solution or other because both solutions haven't any cost with licenses and purchasing.

6.1.5 Maintenance Analysis

The main purpose of the analysis of maintenance is to identify the difference between maintenance costs in Audit Management and maintenance costs in SAP AUD.

As the others solution of the SAP company, SAP AUD is developed with ABAP, a programming language that belong to SAP. Qimonda doesn't own experts for the development in this language, so, it has a contract with an external company for any development in the SAP system.

The three most usual maintenance operations consist in the addition of a new feature to application, the development of a small change to the application and the correction of application's bug. The comparison of costs for each maintenance type concludes the following:

- New feature – the development of a new feature is considerably higher in SAP AUD than in Audit Management because the price of one man-day of the external company is more expensive than one man-day of Qimonda.

- Development of small change – even for small changes, the development is made by consultants of the external company, so, the costs comparison is similar to the adding of a new feature.
- Bug's correction – in this case, SAP AUD has a lower probability to have bugs but if any bug is detected and a patch to correct it doesn't exist, the correction with the bug has to be made by the consultants of the external company.

According to the previous comparison and to the high value of a SAP expert man-day, the maintenance of SAP AUD is significantly more expensive than the maintenance of Audit Management.

6.1.6 Application Flexibility and Usability

Concerning the flexibility of SAP AUD and Audit Management, the SAP AUD application is less flexible due to the following reasons:

- The insertion of new users is very complex and involves several persons in the process.
- To update the permissions of a user, a process similar to the precious has to be done.
- The development of new features, the corrections of bugs or small changes in the application require approvals for several persons and the process become too slow.

However, the SAP AUD solution allows little customizations that can be made by the users of the application. For example, a new audit category may be added to the application without any development.

Concerning the usability of both the applications, SAP AUD has the advantage of the use of templates, that is, a user can create a template for an audit and reuse this template for an audit similar. This feature permits a reduction of the time spent with the creation of the audit.

On the other hand, the numerous features provided by SAP AUD and the attempt to cover the maximum possible of details turns the application complex and difficult to use for beginners. Besides that, the users of SAP solutions frequently refers that the user interface of the SAP systems is not user-friendly.

6.1.7 Conclusions

According to the several factors previously analyzed, the Audit Management application shouldn't be replaced by SAP AUD.

The SAP AUD application is more complete than Audit Management and belongs to a company recognized for the quality of its products. However, the numerous features provided reduced the application usability and make the application more difficult to use. Consequently, it is necessary much time for training the users of the application. Besides that, the features of Audit Management were specifically developed for the usage in Qimonda sites, so, they are adapted to the company needs and represent an important advantage for the application comparatively to SAP AUD.

One of the most important factors is the comparison of costs associated to each application. Both the applications haven't costs for acquisition but SAP AUD has greater maintenance costs than Audit Management and the difference between those costs in considerable.

6.2 Integration with Action Plans tool

6.2.1 Context

Action Plans is a concept that exists in several enterprises. This concept consists of defining a set of actions to correct some non-compliances with quality, security or environment regulations in order to improve some processes or practices. An action plan may have different origins like audits or inspections and it is assigned to one or more people who should achieve the stipulated tasks within the defined deadlines.

Generally, in large enterprises, these plans are supported by software applications due to the need to guarantee the organization among the people involved in it and to maintain records of the performed tasks for later analysis.

In Qimonda, action plans are put into effect in case of non-compliances with regard to quality, environment, hygiene, health and security. Support for these action plans is provided by Klusa however the usage of this application will be discontinued due to its high maintenance costs and it will be replaced by a new application that belongs to an external company and it will be internally called Action Tool. At the present the transition phase between applications is almost achieved and, soon, Action Tool will be used for all action plans in the company.

6.2.2 Objectives

Due do the non-viability of Klusa, it was necessary to find a solution to support the management of action plans. Most of the time, an action plan is the consequence of an audit; therefore it makes sense to analyze the possibility of merging these two features.

The main goal of this analysis is to identify the best way to optimize the interaction between the management of audits and the management of action plans focusing on its usability. The common point between audits and action plans are the tasks. An audit consists of detecting deviations and defining preventive or corrective actions for those deviations. Software solution for management of audits, Audit Management, doesn't include a concept of action plans but it allows the user to add as many tasks as need for each deviation. An action plan is slightly more complex than a simple list of tasks and provides a larger set of features. So, it is

advantageous for the users of Audit Management to make full use of the functionality of management of action plans.

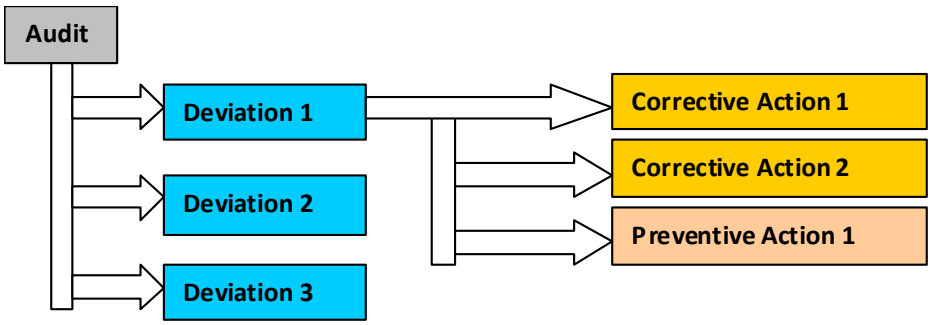


Figure 57 - Relationship among audits, deviations and actions

At the present, if during an audit someone wants to define an action plan he/she has to do it in Action Tool because Audit Management doesn't support that feature, however, at the end of this analysis, one will be able to come to a conclusion as to which is more viable.

To sum up, there are two options:

- To maintain the present situation, this is to maintain management of audits and the management of action plans in different and independent applications.
- To develop in the Audit Management application features to allow the management of action plans.

6.2.3 Action Plans requirements

The requirements for the management of action plans may de divided in three main categories: plan creation, plan follow up and reports. The requirements for each category are listed below.

6.2.3.1 Plan Creation

Characteristics of each plan:

- Plan's type: Quality, Environment and Security.
- Actions with sequential numbering within each type.
- Plan's number.
- Plan's date.

Selection of plan's information:

- Deviation's origin:
  - Quality: audit.
  - Environment: audit, monitoring, accident, incident, external complaint, internal complaint.

  - Security: audit, monitoring, accident, incident.
- Entering the deviation by plan owner or auditor team in the program:
  - Automatic numbering by fiscal year.
  - Classification: non-compliance or observation.
  - Identification of the area where it was detected.
  - Identification of the person who detects the deviation.
  - Validation by plan owner and selection of the action performer.
- Email sending to action performer.
- Daily email to action performer until action performer replies.
- Action performer's response:
  - Record deviation's origin.
  - Suggestion of corrective action and deadline.
  - Validation by direct supervisor (if applicable)
- Email to plan owner with the action performer's response.

6.2.3.2 Plan Follow Up

- Follow up of actions implementation with comment fields.
- Email to the action performer two weeks before the deadline to complete the task expires.
- Closure of actions by plan owner.
- Evaluation of efficiency of the corrective action.
- Reopening of action if the previous results prove to be inefficient and repeat previous steps.
- Layout of action plan with different colors according to action status. (figure 58)

Origem	NC/Obs	Constatação	Causa	Acções correctivas	Área	Responsável	Data prevista de encerramento	Data de encerramento	Verificação
AC 26/01		Audit Council - document non controlled		Control the Document "Audit Council" in the documentation system		QM / Ana Pereira	30-06-2002	05-07-2002	18-07-2002
AC 40/01	Obs	Cracks in the waste water floor		Solve the problem of coating on the wastewater treatment area floor: - Technical study presented until Jul 01 - Decision about solution and establishment of implementation timetable until Oct 01		FM / Fernando Vieira	31-10-2001 30-09-2003		
AC 41/01		No appointment for the Crisis Management Team		Plan Crisis Management & Business Interruption activities. Announce the Crisis Management Team, team leader and meeting room		SS&E / Carlos Lourenço	31-05-2003 31-07-2003 31-10-2003	30-09-2003	

Figure 58 - Layout of actions showing different action status

6.2.3.3 Reports

- Creation of reports on the supervision of the actions (monthly or with other time period) with graphs and text.
- Creation of reports on the overdue actions.

- Calculation of indicators for scorecards:
    - Example 1: percentage of actions completed (quality, environment and security) within the deadline, by the department.
    - Example 2: percentage of quality actions completed within the deadline, by year or trimester.

6.2.4 Application Analysis

6.2.4.1 Action Tool Description

Action Tool is the software application adopted by Qimonda to manage action plans. This solution consists of a web-based application provided by an external company that ensures application hosting and provides exclusive web access to Qimonda users. All the mandatory requirements are covered by the application and other features that improve usability also are available. Thus, this application provides a tool to create action plans, assign actions and follow up on the steps. It has a profile based interaction, with security restrictions, and an email reminder mechanism. Besides these features, Action Tool also allows actions rating, visualization of the percentage of the completed tasks and printout of reports.

Action Tool doesn't depend on other applications and is compatible with all types of action plans used in Qimonda independently of their origin. This analysis aims at action plans that result of audits, which are completely compatible with this application.

Action Tool is not in productive usage at the moment because a user's training phase is under way but its acquisition has already been made. The contract negotiated between companies involved includes clauses for preventive and corrective maintenance which consist in upgrades of technologies or bug's correction.

The source code of the application is not opened and only the company that created the application can make changes in the code or add new features. However Qimonda benefits of a set of working-hours to be used for that purpose. Nevertheless, these hours aren't enough to allow big changes or the addition of new features. The addition of any new feature implies negotiations for more hours between the companies.

6.2.4.2 Audit Management Description

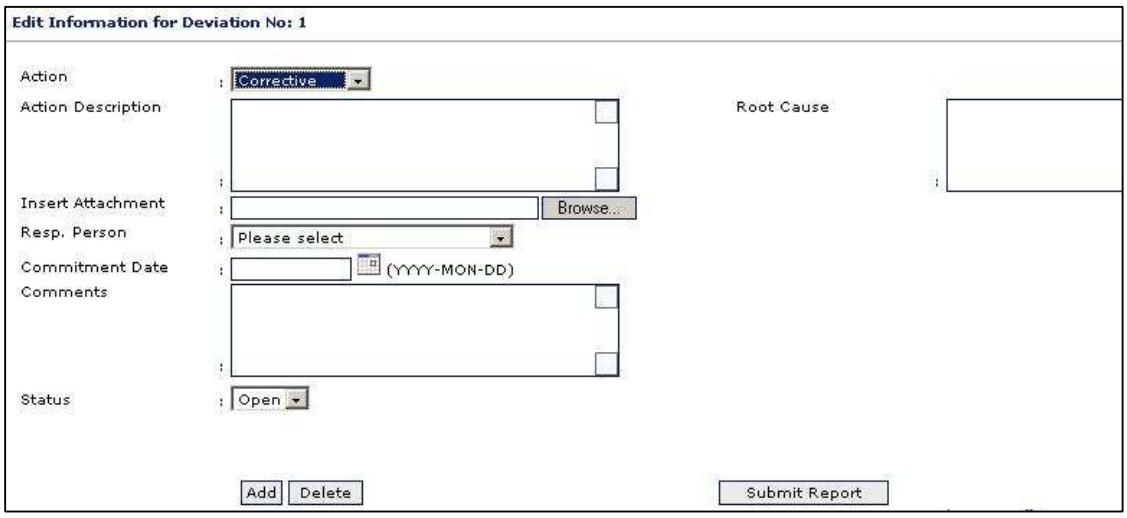


Figure 59 - Add action in Audit Management

Audit Management, an application owned by Qimonda, provides features for the management of audits. The application allows the auditee to add several corrective tasks for each deviation identified by the auditor. These corrective tasks may be compared to a simple action plan.

Moreover, this application also allows one to select customized reports concerning specific actions such as the percentage of overdue actions.

Audit Management has many points in common with the management of action plans which make it possible to enter all the features of the action plans in this application.

6.2.5 Solutions for the integration

6.2.5.1 Insertion of Action Plans' features in the Audit Management application

This solution consists of developing, in Audit Management, all the features required for the management of action plans, which makes this application compatible with other action plans resulting from other sources besides audits. Thus, a specific application won't be needed for the management of action plans.

Action Tool, which has very low costs, is being used at the Qimonda site in Porto at the moment; however the evolved Audit Management application may be used in other sites and may replace local applications for management of audits and management of action plans, which represents a considerable cost reduction for the enterprise.

The development of this solution should follow the roadmap described below.

Implementation Roadmap

The upgrade of Audit Management implies a development effort that represents the biggest cost associated to this solution. Nevertheless, if this solution is adopted, many features and the source code of the application should be reused to reduce development costs. In fact, this consists of a reengineering of the application; its recommended roadmap is described in this section.

The application's name might have to be changed because it will not be exclusive for the management of audits; it will also sustain the management of action plans.

Audit Management is going through a migration phase with respect to technology and programmatic languages. The new application is not in productive usage yet but this roadmap will focus on it because that migration is expected to end soon.

The main steps of this upgrade's implementation are listed below:

1. Prepare database to support action plans. This consists of adding to the existing database a few tables and some foreign keys between tables.
2. Create two icons that allow choosing between management of audits and management of action plans in the first window of the application to pop up.
3. Add a feature for the creation of the action plan. The flow of this functionality is similar to the flow of the creation of a new audit; therefore, it should be taken as an example. In the source code, the file named AuditRequest should be used as base to the implementation of this feature.
4. Reuse the feature of Audit Management for deviations' addition because this concept has the same meaning and the same flow in audits and in action plans. In Audit Management, the deviations are added by the auditor, whose role is similar to plan owner's role, and source code for this functionality is present in the file AuditorQMUpdate.
5. Reuse the feature of action's addition from Audit Management and develop it for its correct usage in action plans
6. Implement a feature for reports generation. Some report types (percentage of the overdue actions, for example) are already implemented in Audit Management, so, they should be reused and should serve as a base for other report types.

Some changes should be done in the application menu but the general structure should be maintained. In a first approach the menu rearrangement should be like the one shown below.

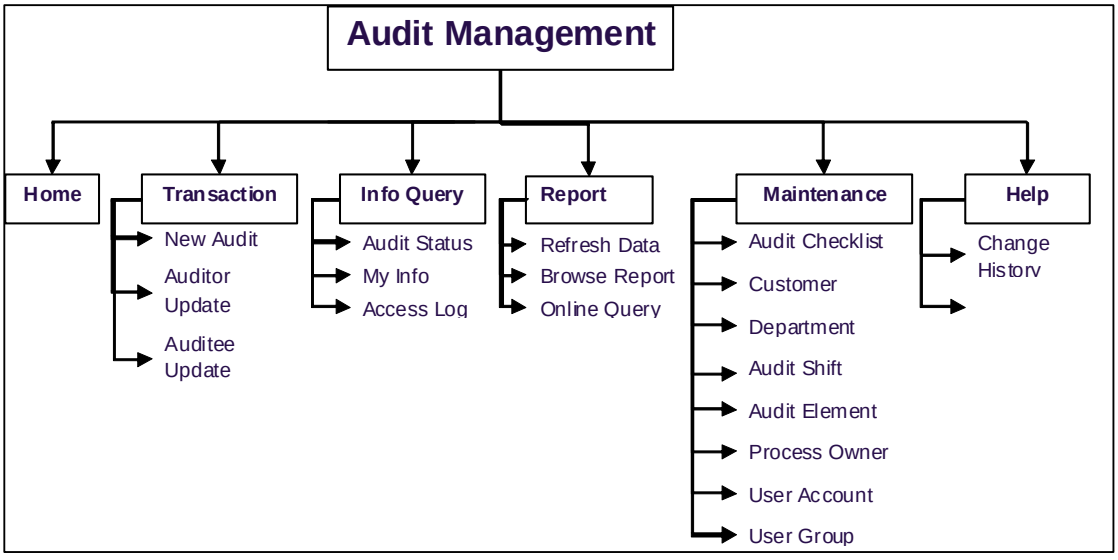


Figure 60 – Current menu of Audit Management

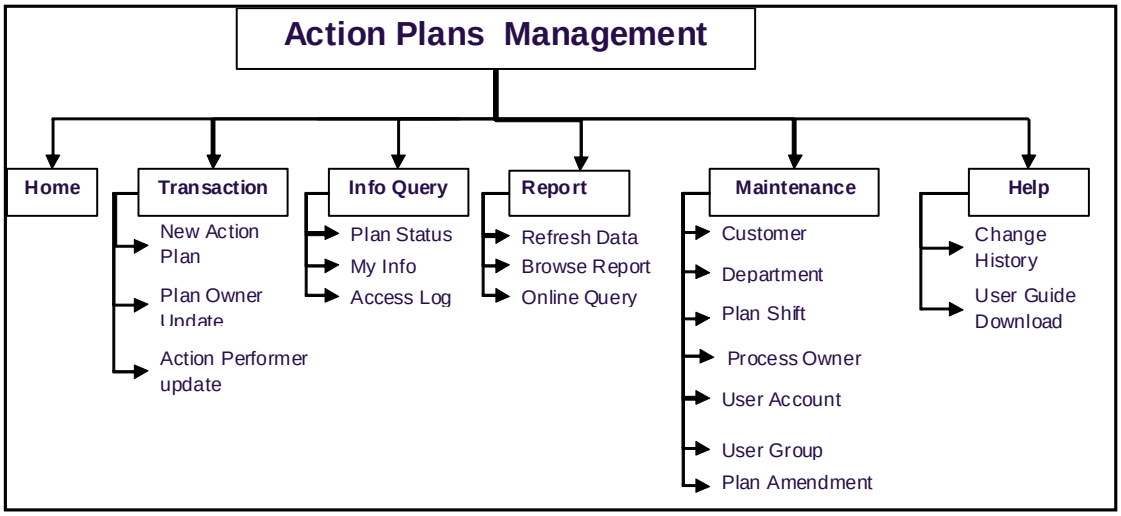


Figure 61 - Application menu with features for action plans

6.2.5.2 Interoperability between Action Tool and Audit Management

This solution consists of maintaining the management of action plans and the management of audits in separated applications. Thus, any action plan resulting of an audit should be inserted in Action Tool while other information about audit should be inserted in Audit Management.

In this case, a little customization should be done in Audit Management to introduce a link for Action Tool in the page where the user adds the deviations.

6.2.5.3 Solutions Comparison

Features

The features required for the management of action plans are provided by Action Tool and are listed in the table below. This table identifies features which are present in Audit Management.

Action Plans Features	Description	Audit Management
Action Plan	Plan Type: Quality, Environment and Security with sequential numbering for each type	no
	Plan number and date, department, start date, end date, attachments, description, etc…	no
	Selection of deviation origin: Quality: audits. Environment: audits, monitoring, accidents, incidents, external complaints, internal complaints. Security: audits, monitoring, accidents, incidents.	no

	Creation of actions	yes
	Creation of sub-actions	no
	Creation of steps	no
	Creation of plans based in other existing plans	no
	Entering the deviation by plan owner or auditor with automatic numbering by fiscal year	yes but without by fiscal year
	Classification: non-compliance or observation	no
	Identification of the area where deviation was detected	no
	Identification of the person who detected the deviation	yes if that person is the auditor
	Validation by plan owner	No
	Assignment of the action performer	Yes
	Action performer's response with the identification of the root cause	Yes
	Corrective action suggestion and action deadline	Yes
	Validation by the direct supervisor (if applicable)	Yes
Roles	Action Owner	yes (auditee role in AM)
	Person responsible for the action	yes (action owner role in AM)
	Process Owner	Yes
	Hierarchy and filtering by category/department	Yes
Notifications	Email to action performer	Yes
	Email to action performer 2 weeks before the deadline to complete the task	No
	Daily email until action performer's response	yes
	Email to plan owner with action performer's response	yes
	Distribution lists	yes
Monitoring	Monitoring of the percentage of the completed tasks, ex: 60%, 70%) and comment fields	no

	Action closure by plan owner	no
	Evaluation of the efficiency of the corrective action	no (it's only possible to say if it's done or not)
	Reopening of the action if it was inefficient and repetition of the previous steps	yes
	Layout of actions with different colors according to the action status	no
	Action tracking (last update, last modification, …)	no
Reports	Creation of reports of actions' monitoring (monthly or with other time period) with graphs and text	yes for some types of reports
	Creation of reports of the overdue actions	yes
	Calculation of scorecard indicators	no
	Example 1: percentage of the actions completed within the deadline by department	no
	Example 2: percentage of quality actions completed within the deadline by year or trimester	no
Others	Allow attachments to action plan	no, but it's possible to add attachments to all actions
	Visualization of plans' ranking	no
	Association between plans	no

Table 5 - features for action plans available in Audit Management

If the decision is to maintain both applications working separately all these features will be available through Action Tool. If the decision is to evolve Audit Management it will be necessary to develop features that are not available and rearrange other features that don't totally comply with the requirements.

Usability and Flexibility

Comparing both solutions it is evident that usability stands out in the first case, this is, if the features for the action plans are integrated in Audit Management. The advantages of this solution are the following:

- All the features are contained in the same application avoiding, thus, the user interaction with two applications.
- The user doesn't have to insert the same data in different applications.
- Avoidance of redundant data.
- Avoidance of the risk of inconsistent data in application's databases.

- Reduction of the time spent.

Concerning flexibility, it's recommended to evolve Audit Management due to the following reasons:

- Application and source code belong to the company simplifying, thus, new developments.
- Small modifications (insertion of a new link, rename of some fields, bug fixing, etc…) may be quickly done.
- Application will be entirely built according to the company needs and requirements.
- Technologies' upgrades will be done at the company avoiding extra bureaucratic processes.

In Action Tool, every modification has to be done by the company that owns the software reducing, thus, the application flexibility and increasing the necessary time to achieve each change.

On the other hand, the inclusion of the management of action plans in Audit Management will slightly augment the application's complexity due to the increase of the available features. Nevertheless, this inconvenience is not enough to annul all the other benefits because Audit Management will not be much different or complex from what it is currently.

Acquisition and maintenance costs

The costs of each solution are the most significant criteria at the time of making a decision. In both cases there are costs associated which are described below.

- Solution 1 – Audit Management upgrade

  Audit Management has no acquisition costs because it already belongs to the company but it has maintenance costs and, if this solution is adopted, it will mean that the company will have to spend a large amount of money to upgrade the application.

  Maintenance costs may be divided in three kinds of maintenance:

  - Preventive maintenance – upgrade of software versions, maintenance of databases, etc…
  - Reactive maintenance – changes' requests, addition of features, etc…
  - Corrective maintenance – bugs fixing.

  There will always be preventive maintenance costs regardless of the solution adopted. However, the probability of the existence of bugs and the need of modifications in the application will increase and lead to more maintenance costs if this solution is adopted.

  To sum up, this solution will add the following expenses:

  - Implementation of features for management of action plans in Audit Management.
  - More corrective maintenance
  - More reactive maintenance

To obtain more concrete values, the costs of the implementation of new features have to be analyzed according to the main development steps identified in a previous chapter. Special attention should be given to reports' generation because that feature may be developed in two ways. The first solution consists of developing programmatically a source code for each report type and the second solution consists of using Business Objects technology. Business Objects is a framework that allows the users to customize all types of reports generated from database values eliminating, thus, any development of a source code.

The estimation of the project costs is made with and without the usage of Business Object (BO) and assuming the following rates for man-day (MD). These are approximated values to the company internal numbers and give a realistic idea of the monetary costs. This estimation don't include maintenance costs, it only considers the project costs until its release.

Function	Rate (Euro/MD)
Programmer	175
Analyst	200

Table 6 – Rates by function

	Cost (MD)	Cost (Euro)
Project specification	10	2000
Implementation		
Database Update	1	
Action plan creation	4	
Deviation addition	3	
Actions addition	7	
Reports generation	10	
Total implementation	25	4375
Application test	6	1050
Total	41	7425

Table 7 - Costs without BO usage

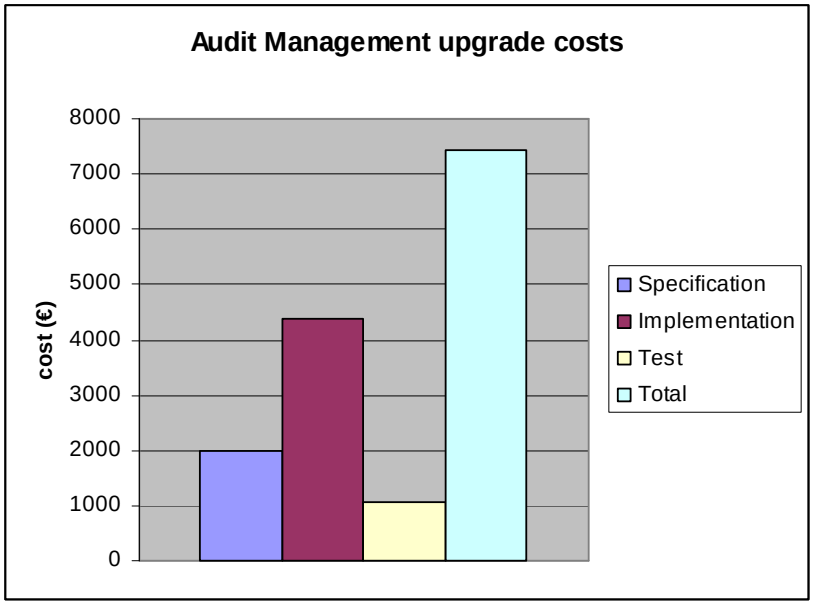


Figure 62 - Graphical view of the costs without BO usage

Project costs with the usage of Business Objects are shown below.

	Cost (MD)	Cost (Euro)
Project specification	10	2000
Implementation		
Database Update	1	
Action plan creation	4	
Deviation addition	3	
Actions addition	7	
Total implementation	15	2625
Application test	6	1050
Total	31	5675

Table 8 - Project costs with BO usage

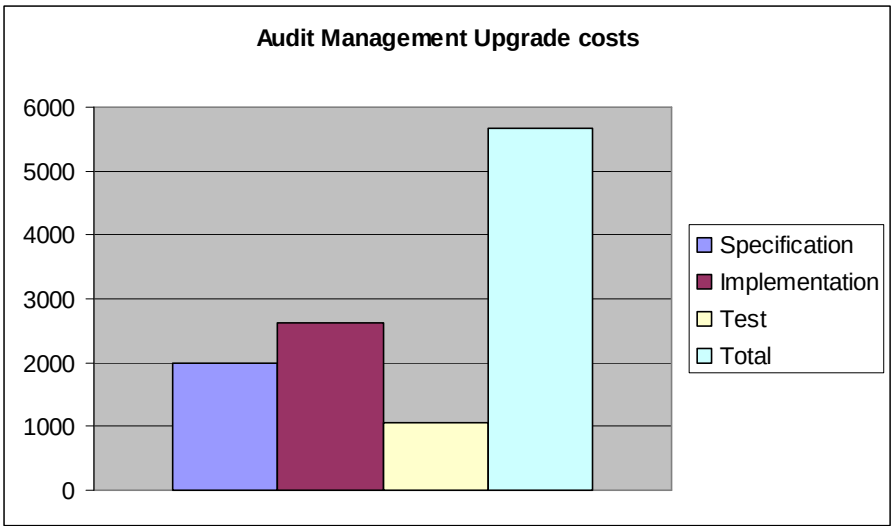


Figure 63 - Graphical view of costs with BO usage

- Solution 2 – Maintain Audit Management and Action Tool

This solution has no acquisition costs but Action Tool is subjected to a monthly payment until the end of the contract. This contract includes a small set of hours that will be spent for corrective maintenance. Corrective maintenance is assured by the application seller and preventive maintenance is also assured by the seller up to 15% of the total application cost.

Should the limit values be exceed, the maintenance has to be negotiated between the companies involved but the Action Tool seller already has a base price for every man-hour.

The graph below shows the estimated costs if the company decides to adopt Action Tool considering that the maintenance costs negotiated in the contract will not be surpassed. It is important to note that the contract duration is of 2 years and the company has to do a constant monthly payment.

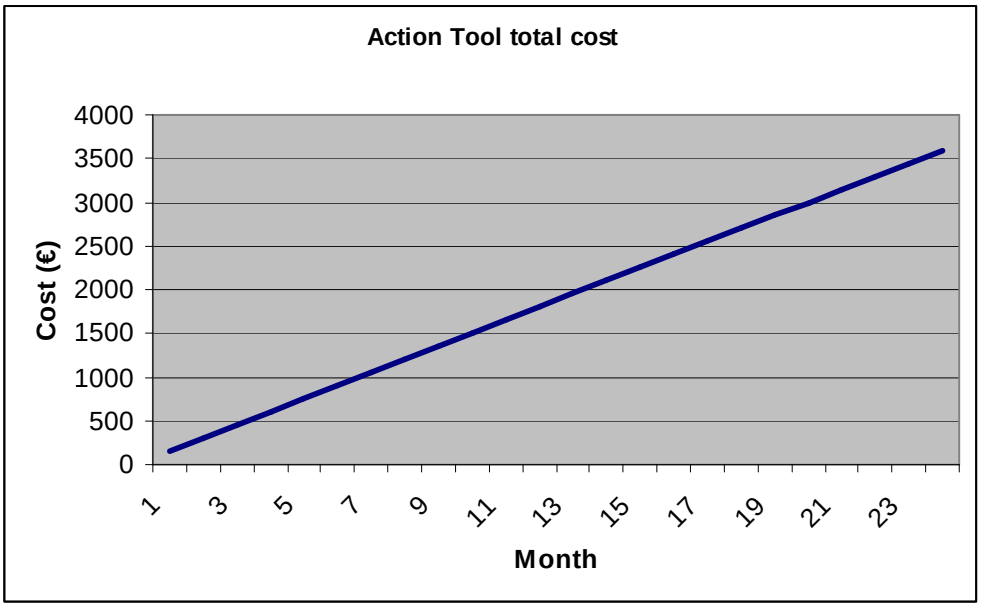


Figure 64 - Evolution of Action Tool total cost

At the end of contract, Action Tool will represent a total cost of €3.600 if there is no additional maintenance cost.

- Costs Comparison

In the graphs below (figures 6, 7, 8 and 9) it's possible to compare the costs of both solutions. The graph in figure 6 displays the costs if Business Objects are not used in Audit Management upgrade while in the graph in figure 8 the costs are calculated according to the Business Objects usage.

The domain of both graphs is 24 months which is the duration of the contract for Action Tool.

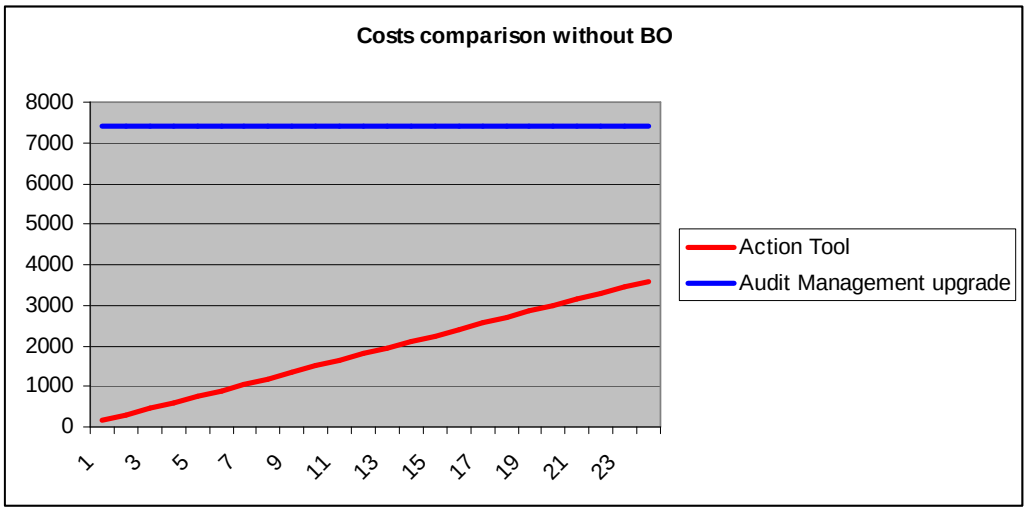


Figure 65 - Costs comparison without BO

As it can be seen in the graph in figure 6, the expenses with Action Tool will not surpass the expenses with Audit Management upgrade during the two-year contract.

The chart below shows how much time is required to justify the investment in Audit Management upgrade.

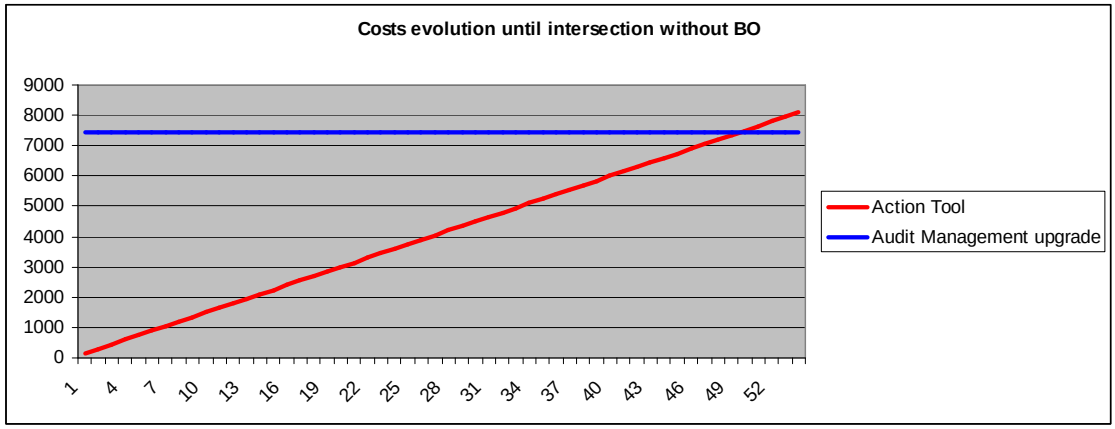


Figure 66 - intersection of the solutions costs without BO

The time necessary until intersection of the two lines as the graph shows is of 50 months which means that the investment in Audit Management upgrade will only be monetarily profitable after 50 months of usage. Bear in mind that these estimations don't consider possible costs of reactive maintenance and it is probable that, within this period of time, some maintenance of this type will be required.

With the usage of Business Objects the difference of the costs between both solutions is minor but Action Tool solution is still cheaper as can be seen in the chart below.

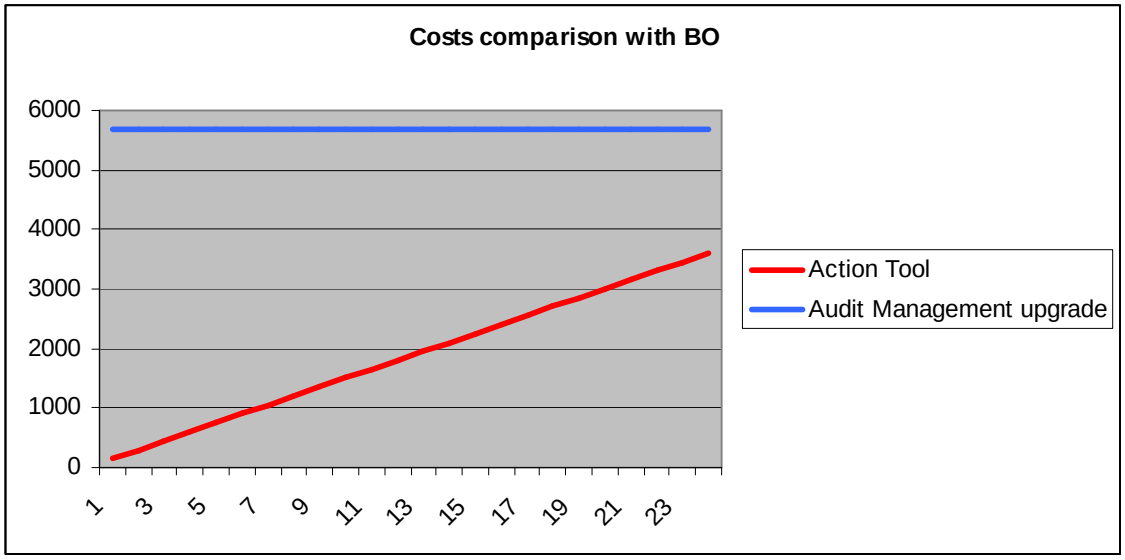


Figure 67 - Costs comparison with BO

To calculate how much time it is needed, the two lines of the previous graph were extended and the result is presented in the chart below.

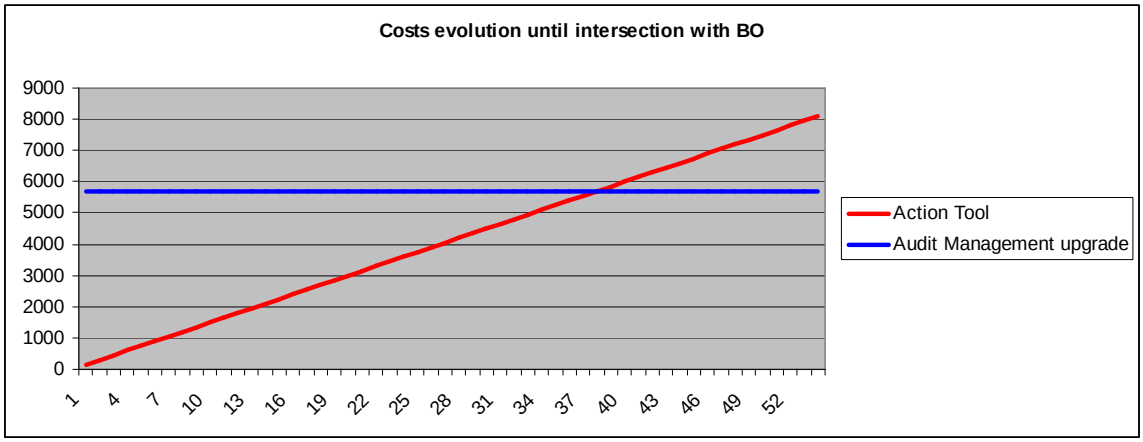


Figure 68 - Intersection of the solutions costs with BO

The calculations came up with the result of 38 months, which means that Audit Management upgrade will only be monetarily profitable after 38 months usage.

To conclude this topic of analysis it's important to refer that these results refer to a single Qimonda site, namely the Porto site, but this solution may be used in all sites and, in that case, costs will be reduced.

The Audit Management upgrade it will not imply more significant costs if the number of sites where it is used increases because the application will already have been implemented and it will only be necessary to install it in a new site. While the option for Action Tool will have additional licensing costs if used in more than one site. However the contract made with the Action Tool seller includes a cost reduction of 20% for every

new license, thus, if this application is installed in other sites other than the Porto site, this new license will be 20% cheaper than the first license.

Below are displayed the total costs for Action Tool usage depending on the number of sites after its usage after two years.

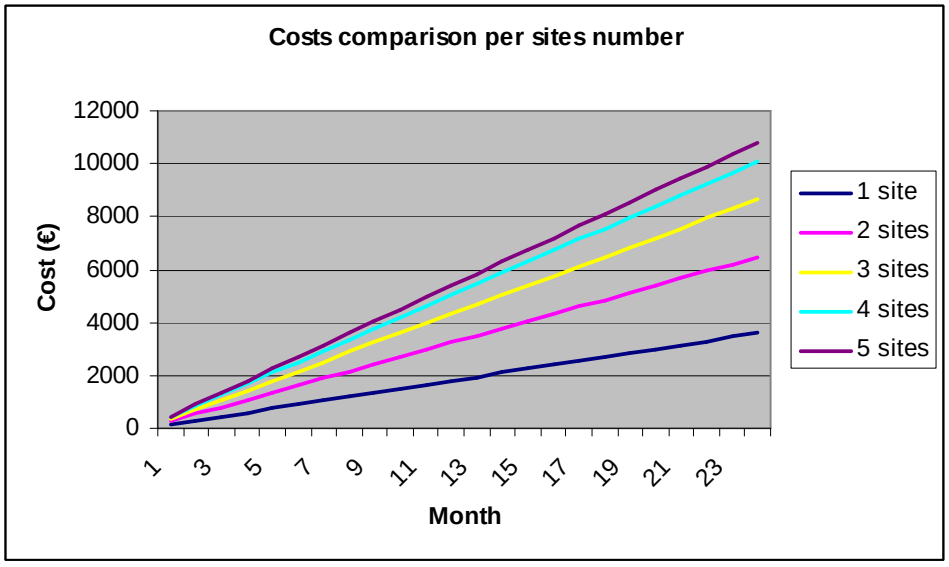


Figure 69 - Total costs for Action Tool per sites number

1 site	2 sites	3 sites	4 sites	5 sites
3600 €	6480 €	8640 €	10080 €	10800 €

Table 9 - Action Tool costs with several licenses

Recalling Audit Management upgrade costs presented previously (€5.675 - figure 8 - with Business Objects usage and €7.425 - figure 6 - without this framework) and the values presented in the above table, it is clear that the Audit Management upgrade is the cheapest choice when used in 3 sites. This solution will even be more profitable if used in 2 sites integrated with Business Objects framework.

6.2.5.4 Results Evaluation

The decision between both solutions presented depends on the importance attributed to the different analysis' factors. The results displayed below are calculated based on giving all the criteria the same importance. So, the person who makes the decision on one of the solutions has to adapt these results according to his/her preferential criteria.

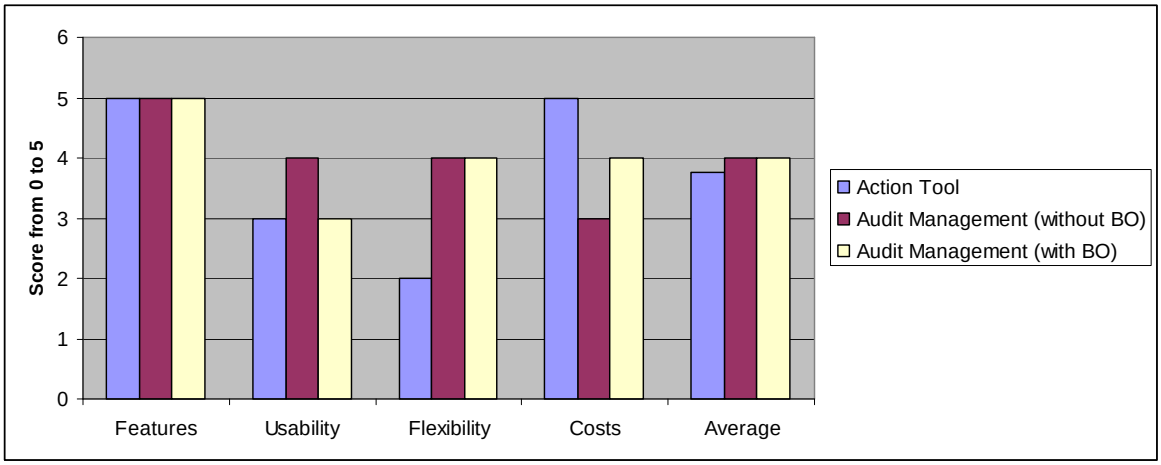


Figure 70 - Solutions comparison criteria

Decision item	Action Tool	Audit Management (without BO)	Audit Management (with BO)
Features	5	5	5
Usability	3	4	3
Flexibility	2	4	4
Costs	5	3	4
Average	3.75	4	4

Table 10 - Criteria classification

6.2.6 Conclusion and Recommendations

This analysis was made based on cost reduction policies and discontinuation of legacy applications. It has been proven that it is possible to add some functionalities to an application when, formerly, these were divided in several applications. In this case, it was shown that it is possible to merge the application of management of audits with the application to management of action plans and the main steps to be followed to merge the applications were also shown.

Essentially, this analysis should be a decisive factor on the best way to provide to the users of the quality department software solutions that allow them to do the management of audits and the management of action plans together. Thus, were taken into account the most relevant factors for this kind of decision.

After evaluating the results, it may be concluded that if the intention of the company is to adopt Porto site solution for other Qimonda sites it is highly recommended to upgrade Audit Management into a more complete application that also provides the management of action plans. Thus, Qimonda will have a robust software application specifically developed for the company and entirely configured to its needs with its inherent advantages. Even if it is only taken into account the usage in Porto site, the option for the Audit Management upgrade should be considered if it' is expected to be used for many years.

On the other hand, it's clear that Action Tool is a cheaper solution and completely focused in action plans management which confers a total coverage to this entrepreneur process. For short duration usage and exclusive to Porto site it's recommended to adopt this solution which frees IT department from any concern with application hosting or hardware maintenance because the application is hosted in the seller's company servers.

6.3 Analysis of reporting mechanisms

In the old Audit Management application the data reports were generated by COM+ components and were presented in Microsoft Excel files. The analysis of the reporting mechanisms was done as a consequence of some requirements of the stakeholders of the project.

The users of the application frequently complain about the rigidity of reporting mechanism and the time needed by the application to perform the operations related to the management of reports. If the application is used in several sites, each site wants to adapt the reports with the values more important for it and this customization is very complicated as the reports are currently implemented. So, the main requirement for the reports generation is its flexibility.

Two software solutions were analyzed to determine to best way to manage the data reports of Audit Management: the usage of Microsoft Excel and the usage of Business Objects.

6.3.1 Microsoft Excel

Microsoft Excel is currently used by Audit Management application. The extraction of the data to create the reports is made by the COM+ components and it is exported to Excel files. Depending on the report type, the data extracted may be presented through diagrams or graphs. Generally, the values used in the creation of the graph are stored in other worksheet.

The usage of Microsoft Excel implies the development of the code for the reports generation and the interaction of this code with the Excel application. So, if any change has to be made to some report type, a new development has to be done.

6.3.2 Business Objects

Business Objects is the Business Intelligence solution most used in the world. This application belonged to a company with the same name but it has recently been bought by the SAP company.

The Business Objects application provides components for the creation of reports according to values present in database tables. This solution doesn't need any development to generate the reports because they are entirely configurable across the web interface of the application. The user only has to select the fields that he wants to include in the report from the database

tables. After that, he may arrange and combine the values as he wants to generate the report he wants.

The application provides a set of options for customization of the reports and its interface is user-friendly, so, the user that interacts with the application hasn't to be a person of the IT department.

6.3.3 Solutions analysis

The analysis of the solutions covers the analysis of several factors important for the decision between the two solutions. The features, the costs of acquisition and maintenance, the flexibility and usability of the applications were analyzed.

The features provided by Microsoft Excel are very limited because they are dependent on the values returned from the COM+ components. With these values, the user may adapt the graphs as he wants. However, if the user needs a value present in the database that isn't returned by the COM+ components he can't include the value in the graph. The Business Objects application allows drawing graphs with any value present in the database tables.

Both the solutions haven't any new acquisition cost if they are adopted because Qimonda has already purchased a license for both. In terms of maintenance costs, the Business Objects application will never have maintenance costs because no development can be made in the application while Microsoft Excel will imply any new development when a change has to be done to any report type.

The comparison of the flexibility of the applications concludes that Business Objects is more flexible because a change to any report is very easy to do while with the Excel there is a development that has to be made. In terms of usability, Excel is more easy to use due to the less features provided by it.

6.3.4 Conclusions

According to the study done about the reporting mechanisms, it is recommended to adopt the usage of Business Objects for the generation of reports in the Audit Management application.

The usage of this solution represents low costs to the company because it doesn't bring any new licensing cost and doesn't have maintenance costs. Besides that, the users of the application practically doesn't need support from the IT department because the application os easy to use. Eventually, any support for the initial configuration has to be done but, after that, the users are completely autonomous to customize the reports as they want.

As Audit Management and Business Objects are two applications web-based, the interaction between them is easy and may be achieved with the inclusion of an hyperlink in the first application. An option for the integration of the application consists of creating a frame in a web page of the first application and includes the web page of Business Objects within the created frame.

Chapter 7

7 Conclusions and Future Work

The existence of a legacy application for the management of audits at Qimonda was the reason that leads to the execution of this project. The application contained several weaknesses like the difficulty of maintenance, the inflexibility of its source code and the weak performance. To maintain the productivity and the effectiveness of the management of the audits, the reengineering of the application was inevitable.

The approach to the execution of the project was based in the study of the state of the art done previously of the specification of the application. In this approach the contextualization of the project in the company's environment was also taken into account. Thus, it was concluded that the usage of software tools to help reengineering the project, particularly converting the source code, doesn't benefit the final results of the project because those tools don't speed up the development process and the code produced is unreadable.

The approach selected to the project's execution starts with the analysis of the old Audit Management application to identify the requirements for the new application. Afterwards, several architectures for the application were analyzed in order to identify the one that best fits in the project requirements with the aim to select that best fit in the project requirements. A prototype with few features to validate the architecture selected was implemented and the remaining features were progressively added to this prototype to complete the implementation of the application. The last phase in the project consisted of analyzing several solutions for the upgrade of the implemented application.

The reengineering project was successfully executed because the functionalities available in the old Audit Management application are still available in the new application and the non-functional requirements were achieved. The accomplishment of this project also provided more experience for future reengineering projects.

In the final phase of the project, after the three analyses of the feasibility of the upgrade solutions for the Audit Management application allowed to conclude the following:

- The integration of features for the management of action plans in Audit Management is recommended if the company intends to use this application in more than one site. Moreover, the usage of the application in several sites is also recommended because it allows a considerable cost reduction for the company.
- The replacement of the Audit Management application by SAP Audit Management is not recommended because the future for the implementation of changes in the application would bring high costs to the company. Besides that, it is recommended to evolve Audit Management to a robust application that provides a large set of functionalities related to the management of audits and the management of action plans and, eventually, also develop other features associated to the quality management.
- The reporting mechanism in the application should be replaced by the usage of a Business Objects solution, which would eliminate the development costs of the

implementation of new report types feature because the reports are entirely configured by the users of the application.

7.1 Objectives accomplishment

The existing problems in the old Audit Management application have been corrected with the execution of this project. Thus, the following objectives were accomplished:

- The maintainability of the application was improved.
- The application was built with the most recent technologies.
- The performance of the application was improved and the time need for the execution of several operations was reduced (authentication, email sending, etc…).
- The flexibility and the modularity required by the stakeholders were improved.
- The application is easier to install.
- The analyses made in the final phase of the project are an important tool to help in the decision about the application's upgrade.

7.2 Future Work

The execution of this project allowed to conclude that the Audit Management application has a strong potential to evolve as it was proved by the analyses made to the possible solutions for the evolution of the application.

The developments that could be made consist of implementing the upgrades suggested by the analyses made in the final phase of the project. Thus, the features for the management of action plans may be added to the Audit Management application and, therefore, the application currently used to the management of action plans has to be replaced. Besides that, the replacement of the reporting mechanism by a Business Objects solution should be done.

The analysis of the inclusion of other functionalities related to quality management in the Audit Management application can be done and, progressively, the application will become more complete and can be widely used by people of the quality department in the Qimonda sites worldwide. Thus, the costs of maintenance of other applications currently used would be avoided and a considerable cost reduction would be achieved.

Chapter 8

8 References & Bibliography

[Bel03] Mounia Belmamoune. A reengineered publication model. November 2003. ftp://cs.pitt.edu/chang/handbook/18.pdf

[BM98] Elisa L.A. Baniassad, Gail C. Murphy. Conceptual Module Querying for Software Reengineering. 1998. http://www.cs.ubc.ca/labs/se/papers/1998/icse20-cm.pdf

[Cur02] Sébastien Curutchet. De VB6 à .NET. December 2002. http://dotnet.developpez.com/articles/migration/vb6_vers_dotnet/

[DDN00] Dr. S. Demeyer, Dr. S. Ducasse, Prof. Dr. O. Nierstrasz. Object-Oriented Software Reengineering. February 2000. http://www.iam.unibe.ch/~scg/Archive/Lectures/OOSR-W99.pdf

[LC97] Paul Lomax, Rogers Cadenhead. Converting a visual basic application for the web. 1997. http://docs.rinet.ru/ActiveVB/ch22.htm. Last access in April 2008.

[Mul06] Alex Müller. Reengineering of a ticket ordering system. June 2006. http://seal.ifi.uzh.ch/fileadmin/User_Filemount/Project_Plans/Mueller_project_plan.pdf. Last access in March 2008.

[Pat03] Kamal Patel. Migrating VB 6.0 to .NET. 2003. http://www.ellkay.com/Migration.htm. Last access in April 2008.

[Ros96] Dr. Linda H. Rosenberg. Software Re-engineering, October 1996. In http://satc.gsfc.nasa.gov/support/reengrpt.PDF. Last access in June 2008.

[SAP08] SAP. SAP Knowledge Warehouse. http://help.sap.com/saphelp_auditmgmt/helpdata/EN/index.htm.

[Som00] Ian Sommerville. Software Engineering, 6th edition, Chapter 26: *Legacy Systems*, Chapter 28: *Software Reengineering*, 2000. In www.comp.lancs.ac.uk/computing/resources/IanS/SE6/Slides/index.html, Last access in June 2008.

[SRP03] Herriot-Watt University, Edinburgh. Systems Reengineering Patterns. January 2003. http://homepages.inf.ed.ac.uk/perdita/Reengineering/. Last access in April 2008.

Faculdade de Engenharia da Universidade do Porto

Reengineering of a Qimonda Wide Audit Management Application

Carlos Miguel Amorim Sousa

Appendices of the Report of Project
Master in Informatics and Computing Engineering

Supervisor: Prof. Miguel Pimenta Monteiro

2008, July

Appendix 1

Audit Management application requirements specification

Requirements Specification

V. 1.0

Document management

Document history

Version	Status	Date	Responsible	Reason for change
0.1	in progress	2008–03–05	Carlos Sousa	Document creation

Distribution:

Name (in alphabetical order)	**Department**	**Location**
Adélio Fernandes	QPT IT PA DQ	Porto
Alberto Pereira	QPT IT PA AS	Porto
Carlos Sousa	QPT IT PA AS	Porto

Change authorization

Name	**Department**	**Location**
Alberto Pereira	QPT IT PA AS	Porto

Authoring tools

MS Word 2003

MS Visio 2003

Contents

1 Overview

1.1 Purpose of the product

The Audit Management application is a software tool used by the Qimonda quality department that replaces the manual way of tracking the audit process. This application keeps a record of all the audits, with all the relevant information, and provides online auto reporting improving, thus, visibility and control on the audit actions. It helps, also, to ensure timely responses to actions to put in place.

It already exists an Audit Management application that is built in an obsolete technology which can compromise the application maintenance and upgrade. Moreover, even if the basic principle of audits is the same in every Qimonda sites, there are some details that depend on the factory where is performed the audit. So, the application shall have enough flexibility to be used in any Qimonda site and, for that, it is necessary to upgrade the architecture and technologies to the latest guidelines standards.

The main objective of this project is to investigate possible strategies for the application migration that includes the redefinition of the application architecture, analysis of some details of implementation like authentication or email notification. It should, also, be done a study about some technologies that can be integrated with the application to verify if they bring some benefits in terms of performance, usability or flexibility. To put in practice all the study done, it should be implemented the application migration.

1.2 Areas of application, system integration and delimitation

The Audit Management application is currently in productive usage at Qimonda site in Malacca (Malaysia) and, at short term, it will be used at Qimonda sites in Porto (Portugal) and Suzhou (China). As this application is used across the web, the final objective is that all the Qimonda sites use this application for managing the audits performed in all Qimonda sites.

The integration with some systems of the existing environment and the integration with other software solutions will be object of study during the project development. However, the actual Audit Management application is integrated with the existing environment, so there is no restriction to do the same kind of integration. For other kinds of integration, it will be done a study to analyze if they are feasible or not.

1.3 Users

Audit Management application may have some types of users, depending on the type of audit. The main user group belongs to the quality department and has the auditor role in the audit, that is, they are responsible for the audit execution. A particular type of audit allows auditor role to be taken by an external person that not belongs to Qimonda.

The persons that are responsible for the area, section or process of the audit have, also, a role in the application and may belong to any department.

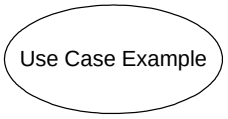
There is another type of user that is responsible for the administration of the application and can configure some details of it.

2 Product requirements

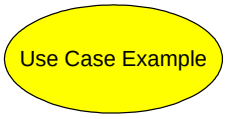
2.1 Functional overview

The Audit Management functionalities are described in the next chapter recurring at UML use cases. A distinction should be done among the use cases that have a yellow background and the other use cases.

The use cases which have a white background correspond to the use cases which are already implemented in the actual Audit Management application. These ones will be implemented in the new Audit Management application.



The use cases which have a yellow background correspond to the use cases which are not implemented in the actual application but they had appeared in the requirements analysis phase like helpful for the application usage. However, these use cases will not be implemented but they will be considered for a later analysis involving the quality department members and the end users of the application.



2.2 System flows

Depending on the audit categories, the audit workflows may be different but all of them involve 3 types of users: auditor, auditee and action owner. In the chapters below, we can see activity diagrams in UML that describe the audit workflows.

In each diagram the states of the audit are identified but there are other states that can't be identified in the diagrams. These states are:

- **Report response overdue:** response to the preliminary report is overdue by the auditee.
- **Action overdue:** the auditee defines a commitment date to achieve the action. If the action owner doesn't perform the action until this date, the action will take "action overdue" status.
- **New action deadline:** If the auditor puts the action status as "Not done", the auditee should define a new commitment date for the action.

We can, also, see the state diagrams with the different states that an action may have.

2.2.1 Action States

To intend better the states that the actions may have, the state diagrams in UML presented below contains all the states and transitions between them. The first one represents what actual Audit Management application allows but, in this case, Action owner doesn't interact directly with the application. The second one represents an improvement possibility that includes the Action owner in the application users.

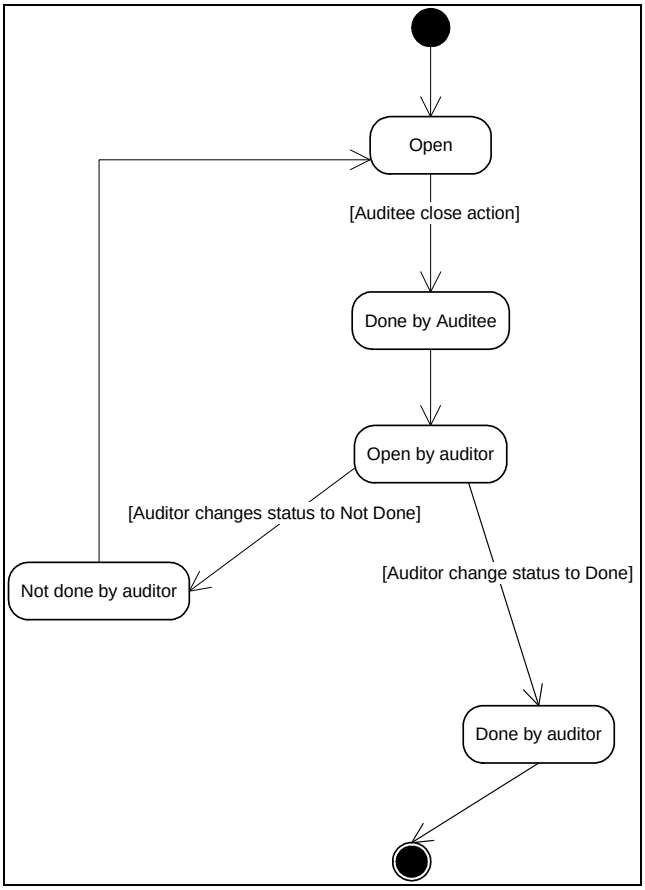


Figure 1 - Action state diagram in actual application

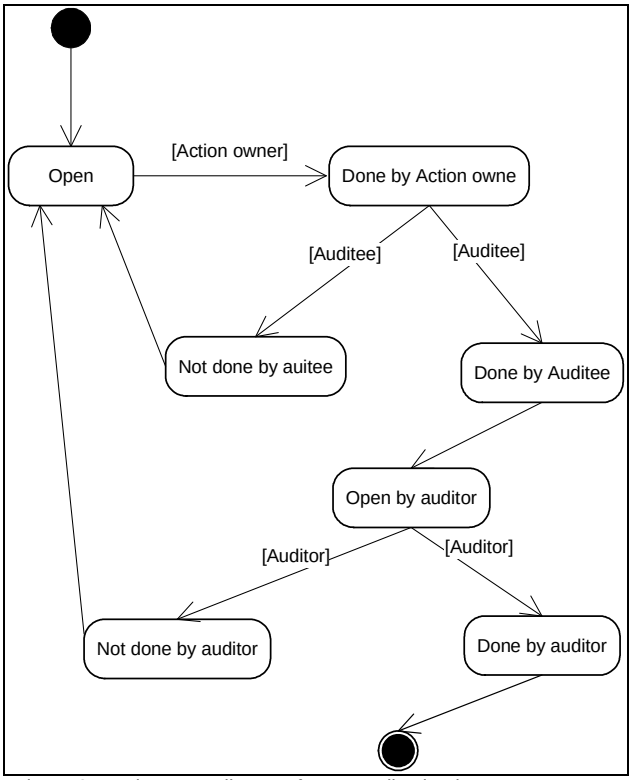
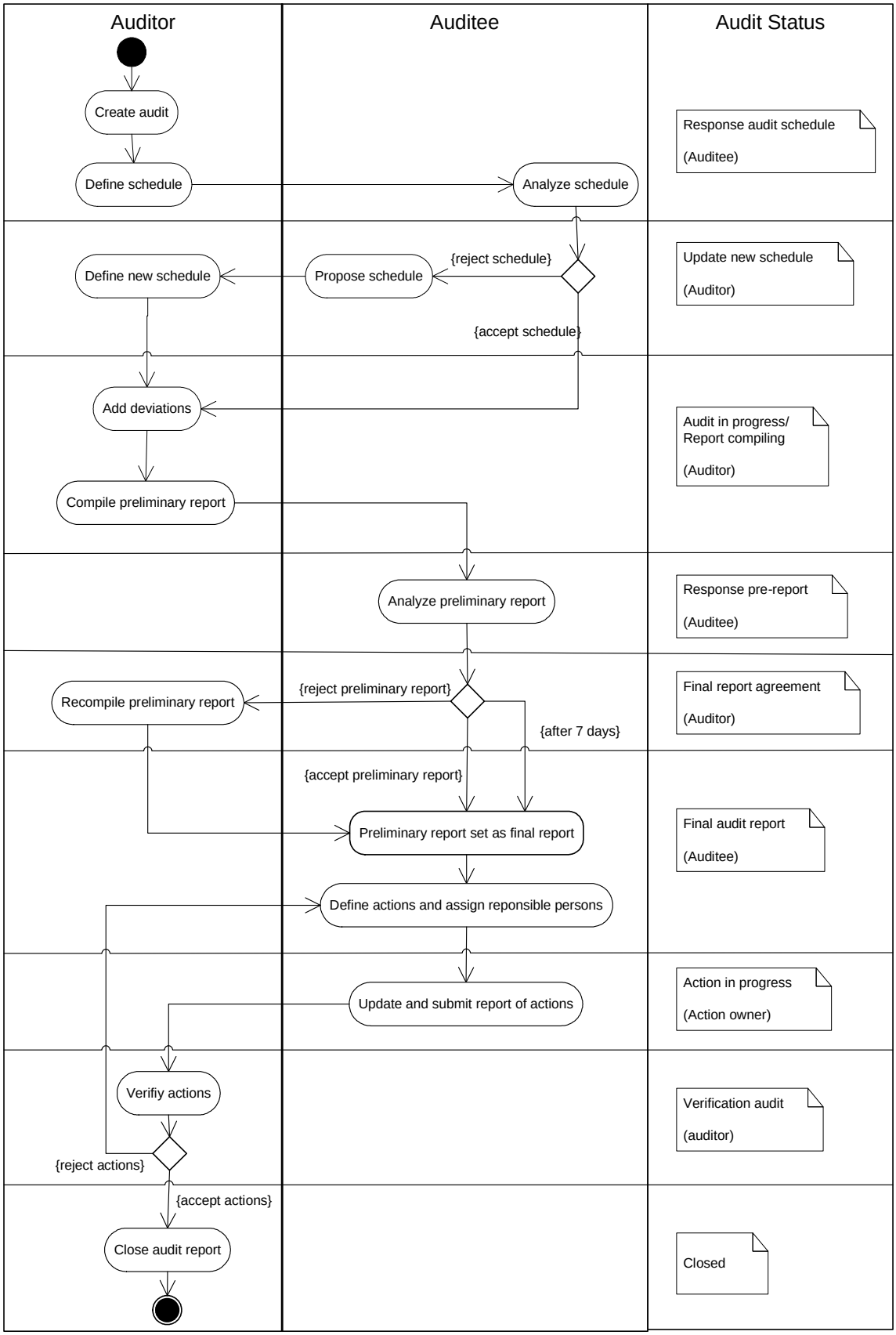
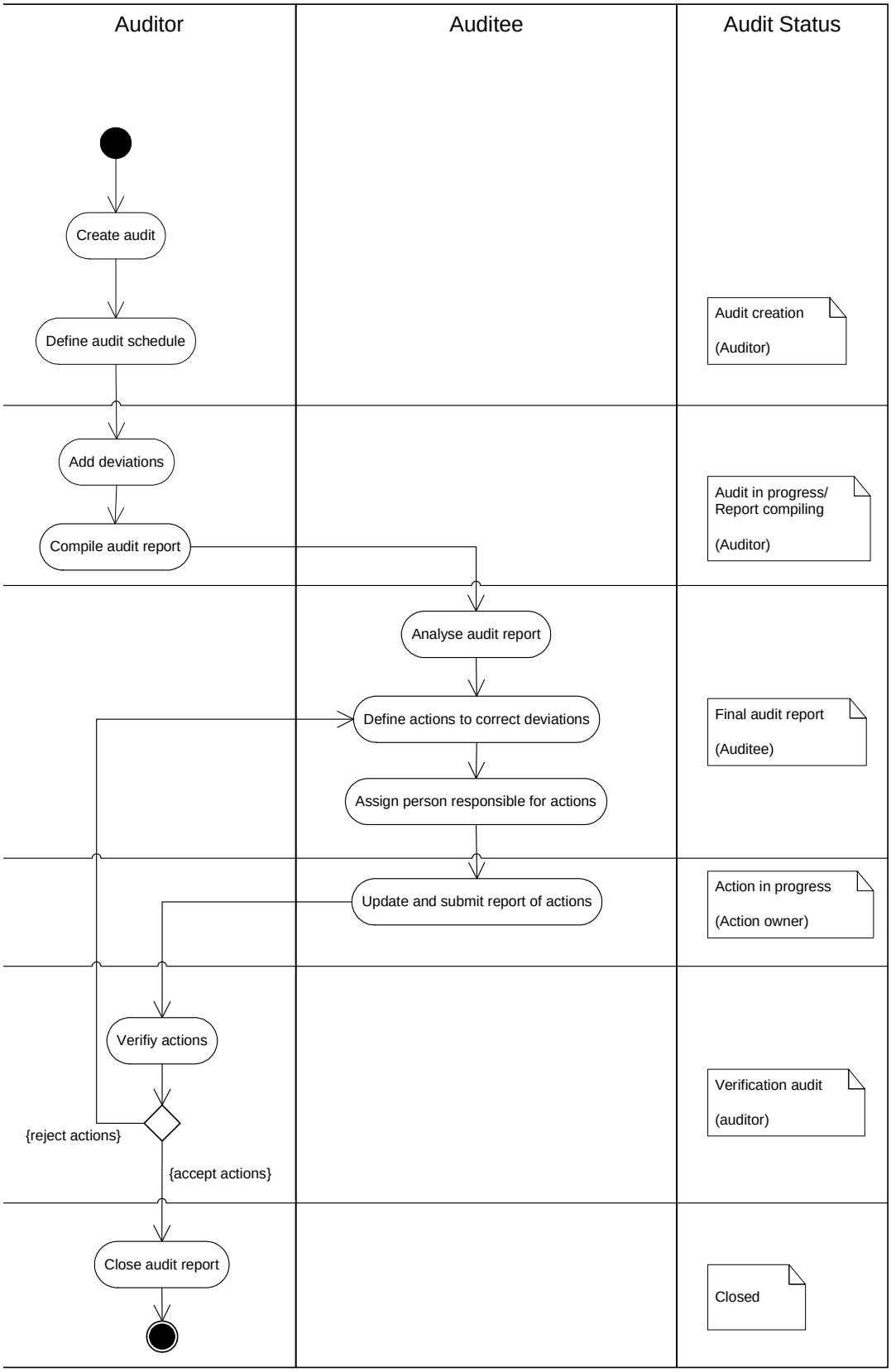


Figure 2 - Action state diagram for an application improvement

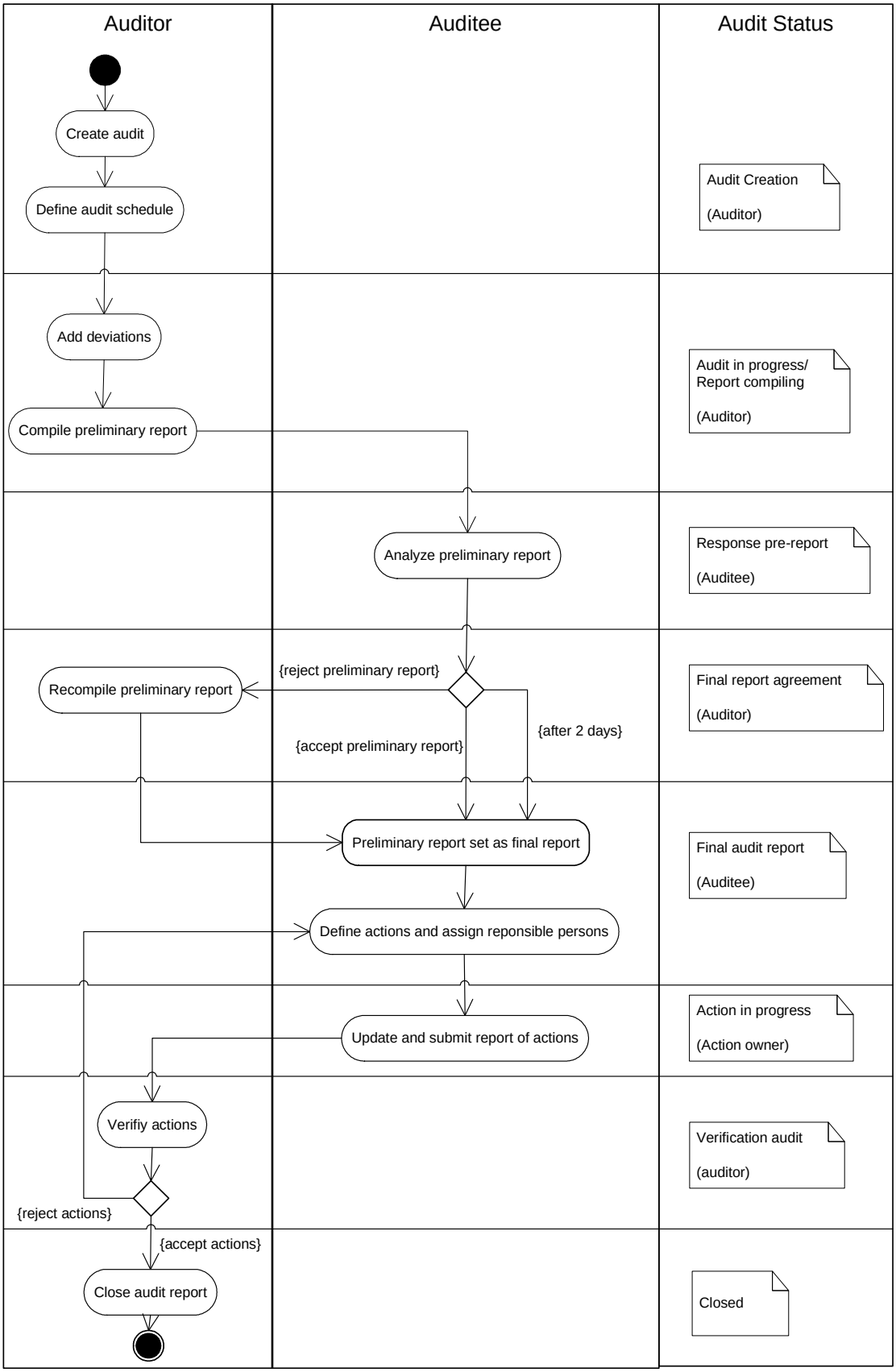
2.2.2 Process or functional audits



2.2.3 General or external audits



2.2.4 Line audits



2.3 Functional description

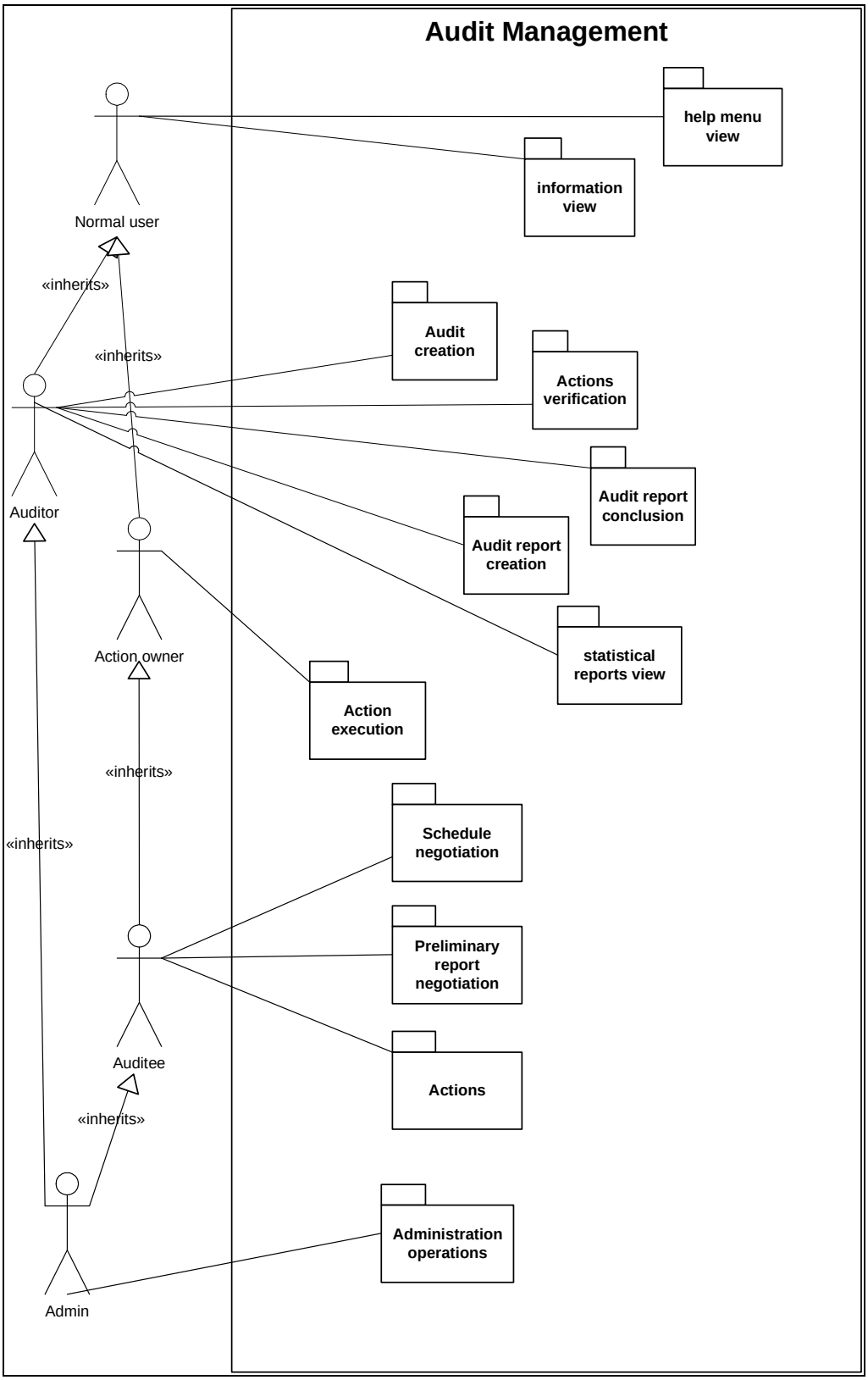
2.3.1 Actors

The Audit Management application will have 5 types of actors which will interact with the system:

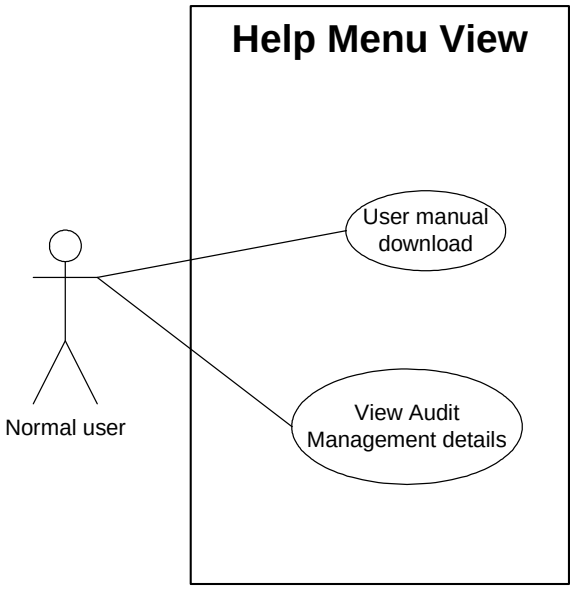
- **Normal user:** this actor has read-only permissions and regroups all the users which can authenticate in the application.
- **Auditor:** this user is responsible for performing the audit. It may be a quality department member or a representative from a customer. He is, also, responsible to follow-up actions that are executed to correct the deviations.
- **Auditee:** this role is played by people that are responsible for processes or functions that are audited. The auditee has to indicate actions to correct the deviations detected by the auditor.
- **Action owner:** This is the user that has to perform actions indicated by the auditee to correct the deviations.
- **Administrator:** this user has the same permissions than other users but he can also configure some details of the application.

2.3.2 Use cases overview

The use cases of the application are grouped by packages. In the figure below, it is possible to see the relationships among the users and the use cases packages.



2.3.3 Help menu view package



2.3.3.1 Use Case «User manual download»

Description	There are some user manuals for the application that may be downloaded by all users.
Scope	Audit Management help menu.
Actor(s)	Normal user, auditor, auditee, action owner and administrator.
Goal	View usage guidelines of Audit Management application depending on the type of user.
Trigger	User action.
Frequency*	Depends on the users actions but it's predictable that it will be rarely used.

Preconditions

1. User is logged in the system.

Minimal Postconditions

Success Postconditions

1. User manual is downloaded.

Main Flow

1. The Normal User accesses to Audit Management help menu.
2. The Normal User chooses the user manual he wants to view.
3. The user manual is downloaded.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface:

Additional Information*

Open Issues*

1. All user manuals accessible for Normal User.

2.3.3.2 Use Case «View Audit Management details»

Description	View details about Audit Management application like version, copyright and so on.
Scope	Audit Management help menu.
Actor(s)	Normal user, auditor, auditee, action owner and administrator.
Goal	Get some details about Audit Management application
Trigger	User action.
Frequency*	Depends on the users actions but it's predictable that it will be rarely used.

Preconditions

1. User is logged in the system.

Minimal Postconditions

Success Postconditions

1. A page with Audit Management details is displayed.

Main Flow

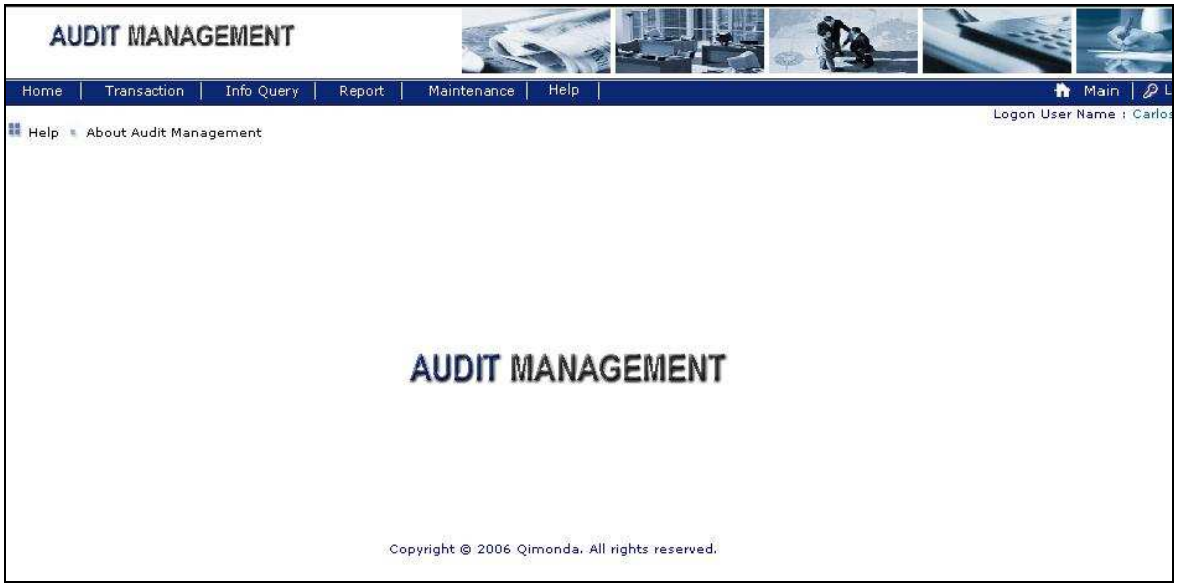
1. The Normal User accesses to the Audit Management help menu.
2. The Normal User selects "About Audit Management" option.

3. Details page is displayed.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface:

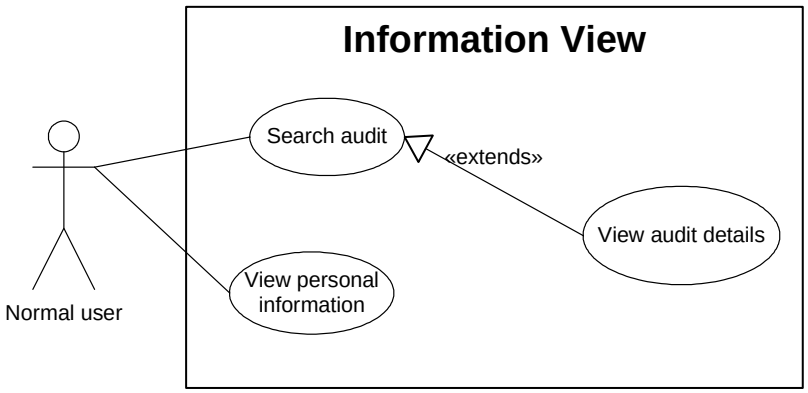


Additional Information*

Open Issues*

1. What information can also be included in this page?

2.3.4 Information view package



2.3.4.1 Use Case «Search audit»

Description	Search audits applying some filters.
Scope	Audit search.
Actor(s)	Normal user, auditor, auditee, action owner and administrator.
Goal	Find one or more audits that match search criteria.
Trigger	User action.
Frequency*	Depends on the users actions but it's predictable that it will be often used.

Preconditions

1. User is logged in the system.

Minimal Postconditions

Success Postconditions

1. A list with summary information of all audits that match search criteria is displayed. Each line of list has a link to specific audit page.
2. An empty list is displayed if there is no audit that matches search criteria.

Main Flow

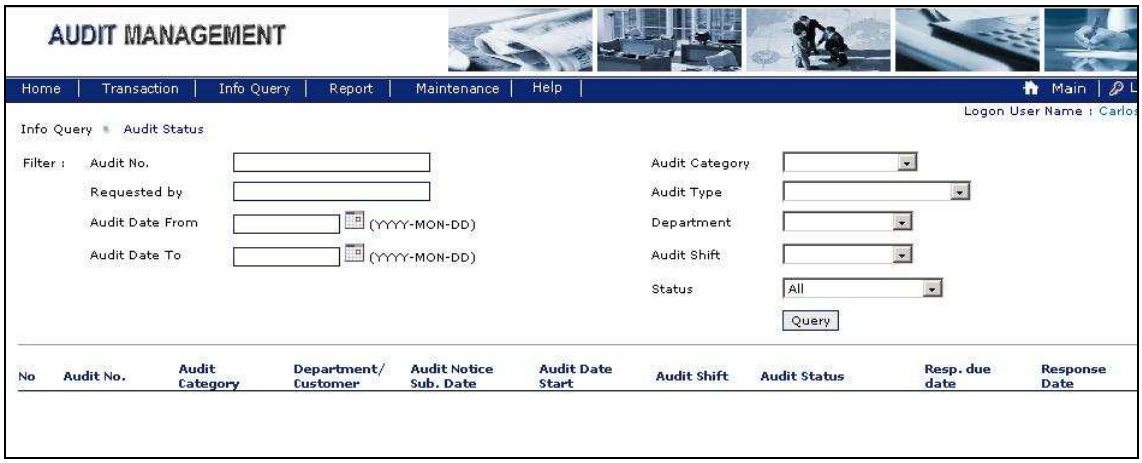
1. The Normal User accesses to Audit Management "Info Query" menu.
2. The Normal User fills search fields.
3. The Normal User executes search query.
4. A list with audits that match the query is displayed.

Alternative Flows / Exceptions*

1. The Normal User accesses to Audit Management "Info Query" menu.
2. The Normal User leaves search fields empty.
3. The Normal User executes search query.
4. A list with all the audits is displayed.

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface before execute the query:



2. User interface after executing the query:

Additional Information*

Open Issues*

2.3.4.2 Use Case «View audit details»

Description	View all the information about one specific audit.
Scope	Audit information.
Actor(s)	Normal user, auditor, auditee, action owner and administrator.
Goal	View audit page which have all the details about the audit.
Trigger	User action.
Frequency*	Depends on the users actions but it's predictable that it will be often used.

Preconditions

1. User is logged in the system.

Minimal Postconditions

Success Postconditions

1. A page with all the details about one audit is displayed.

Main Flow

1. The Normal User accesses audit page through search page.
2. A page with audit details is displayed.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface of one audit page:

Additional Information*

Open Issues*

2.3.4.3 Use Case «View personal information»

Description	View details of logged on user like email, username or domain.
Scope	Information view
Actor(s)	Normal user, auditor, auditee, action owner and administrator.
Goal	User can view its user page for consult some details about itself.
Trigger	User action.
Frequency*	Depends on the users actions but it's predictable that it will be rarely used.

Preconditions

1. User is logged in the system.

Minimal Postconditions

Success Postconditions

1. A page with details about logged on user is displayed.

Main Flow

1. The Normal User accesses Info Query menu.
2. The Normal User selects "My info" option.
3. A page with information about the user is displayed.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

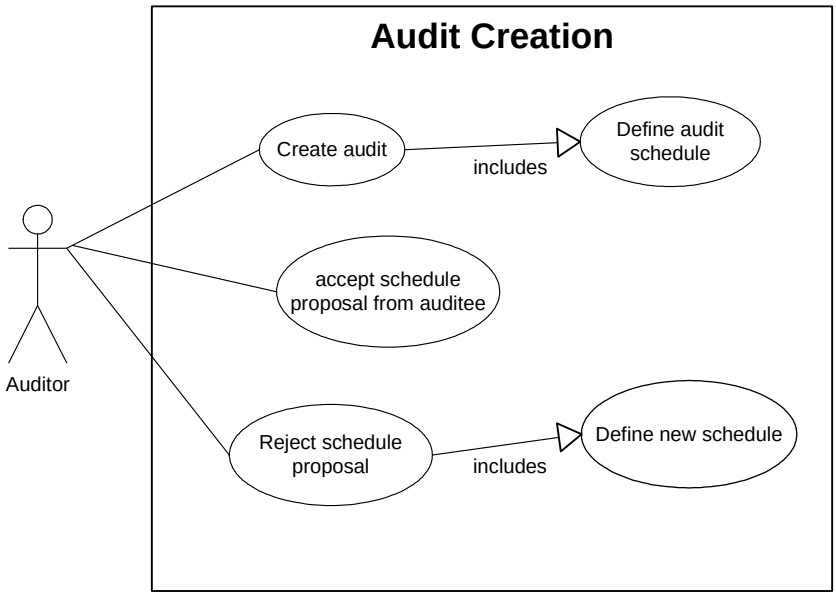
1. User interface of one user page:

Additional Information*

Open Issues*

1. What information can also be included in this page?

2.3.5 Audit creation package



2.3.5.1 Use Case «Create audit»

Description	Request of a new audit that will trigger all the processes needed to audit execution. It's needed to insert some information about the audit that will be performed.
Scope	Audit creation.
Actor(s)	Auditor and administrator.
Goal	Create a new audit whose information will be stored in the Audit Management application providing, thus, the tracking of audit.
Trigger	User action.
Frequency*	Depends on the audits frequency but it's predictable that it will be regularly used.

Preconditions

1. User is logged in the system and has auditor permissions.

Minimal Postconditions

1. A message indicating the success of the audit creation is displayed.

Success Postconditions

1. An email is sent to auditee with a description of audit and a schedule proposal.

Main Flow

1. Auditor accesses “New Audit” page.
2. Auditor selects audit category.
3. Auditor plans audit date and time.
4. Auditor selects audit shift.
5. Auditor selects department where will be achieved the audit.
6. Auditor submits data and an email is automatically sent to auditee.

Alternative Flows / Exceptions*

In case of general audits:

1. Auditor accesses “New Audit” page.
2. Auditor selects audit category.
3. Auditor selects audit type.
4. Auditor plans audit date.
5. Auditor selects department where will be achieved the audit.
6. Auditor selects process owner.
7. Auditor submits data and an email is automatically sent to auditee.

In case of external audits:

1. Auditor accesses “New Audit” page.
2. Auditor selects audit category.
3. Auditor selects audit type.
4. Auditor plans audit date.
5. Auditor selects customer that requested the audit.
6. Auditor selects response due date.
7. Auditor selects process owner.
8. Auditor submits data and is redirected to the page where he may add deviations to the audit.

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface of audit creation page:

2. This use case needs an email sender to notify auditees.

Additional Information*

Open Issues*

2.3.5.2 Use Case «Define schedule»

Description	Define date and time when audit will be performed. In process audits and functional audits this schedule may be negotiated with the auditee.
Scope	Audit creation.
Actor(s)	Auditor and administrator.
Goal	In general audits, external audits or line audits, the goal of this use case is to inform when the audit will take place. In process audits or functional audits the goal is to ask the auditee if he agrees with the schedule proposed.
Trigger	Audit creation by auditor.
Frequency*	Depends on the audits frequency but it's predictable that it will be regularly used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. User is creating a new audit.

Minimal Postconditions

Success Postconditions

1. The date proposed by the auditor should be greater than actual date.

Main Flow

1. Auditor accesses “New Audit” page.
2. Auditor selects audit date.
3. Auditor selects audit time.

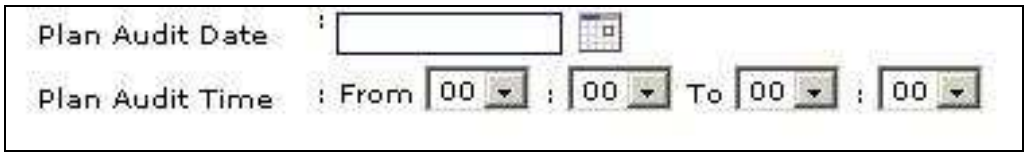
Alternative Flows / Exceptions*

In case of general audits or external audits:

1. Auditor accesses “New Audit” page.
2. Auditor selects audit date.
3. Auditor selects audit time.

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to define audit schedule:



2. This functionality should be easy to use.

Additional Information*

Open Issues*

2.3.5.3 Use Case «Accept schedule proposal from auditee»

Description	After a first schedule proposal, the auditee can accept or reject the proposal. If he reject, he can propose a new schedule that can be accepted or not by the auditor.
Scope	Audit creation.
Actor(s)	Auditor and administrator.
Goal	Accept the audit schedule proposed by the auditee.
Trigger	Rejection of the first audit schedule proposal and a suggestion by the auditee for a new audit schedule.
Frequency*	Depends on the audits frequency but it’s predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditor permissions.

2. A first audit schedule proposal was rejected by the auditee.
3. Auditee suggests a new audit schedule.

Minimal Postconditions

Success Postconditions

1. An email should be sent to the auditee indicating that the auditor has accepted schedule proposal.

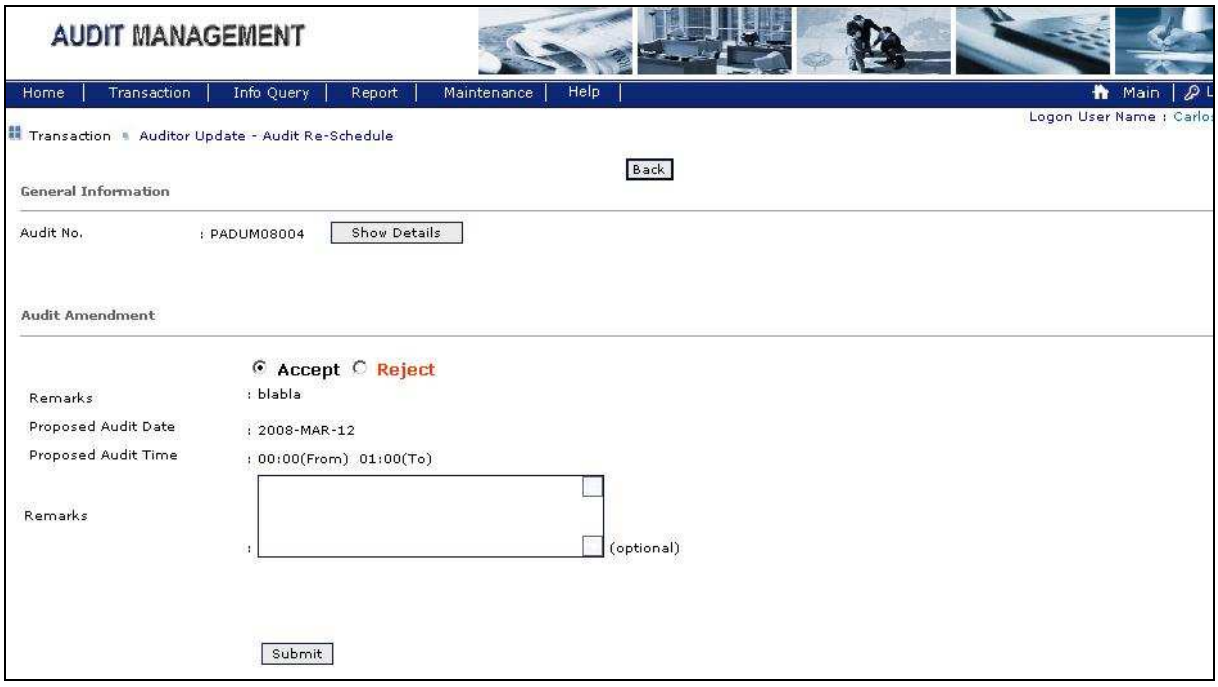
Main Flow

1. Auditor accesses audit page.
2. A warning indicating that auditee has proposed a new schedule is displayed.
3. Auditor accepts new schedule proposal.
4. An email is sent to auditee indicating that its proposal was accepted.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to accept schedule proposal:



2.3.5.4 Use Case «Reject schedule proposal from auditee»

Description After a first schedule proposal, auditee can accept or reject the proposal. If

	he rejects, he can propose a new schedule that can be accepted or not by the auditor. If the auditor rejects the auditee proposal he has to define a new definitive schedule for the audit.
Scope	Schedule negotiating.
Actor(s)	Auditor and administrator.
Goal	Reject the audit schedule proposed by the auditee.
Trigger	Rejection of first audit schedule proposal and a suggestion by the auditee for a new audit schedule.
Frequency*	Depends on the audits frequency but it's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. A first audit schedule proposal was rejected by the auditee.
3. Auditee suggests a new audit schedule.

Minimal Postconditions

Success Postconditions

1. An email should be sent to the auditee indicating that the auditor rejected schedule proposal.

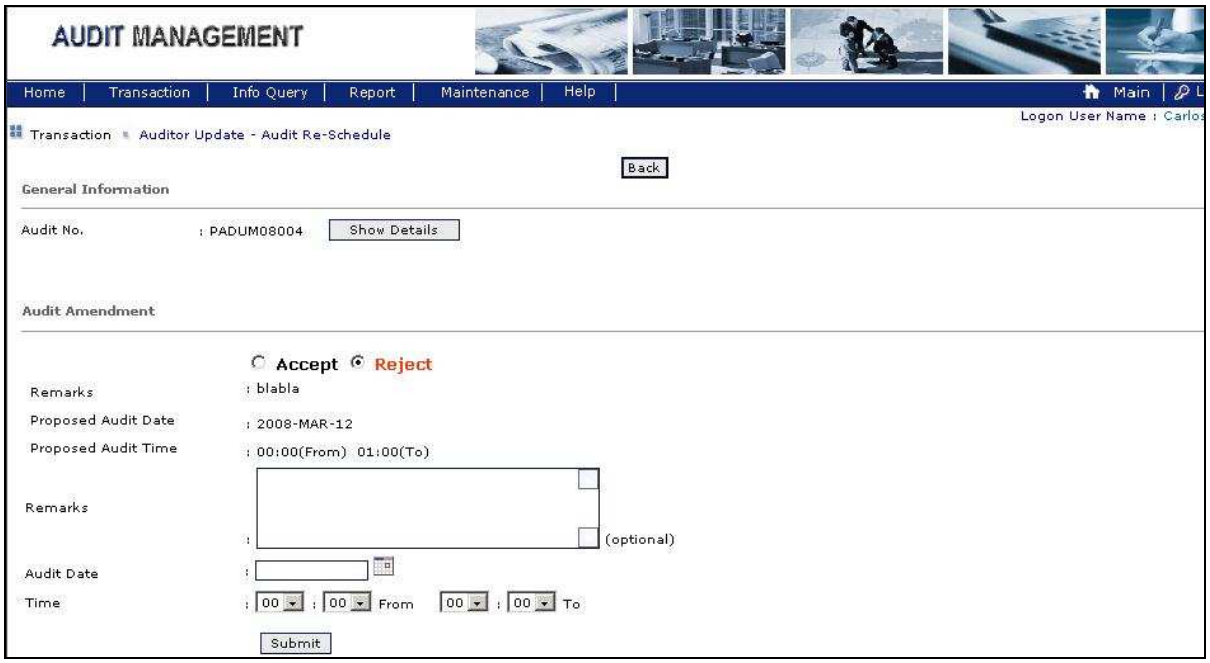
Main Flow

1. Auditor accesses audit page.
2. A warning indicating that auditee has proposed a new schedule is displayed.
3. Auditor rejects new schedule proposal and define a definitive audit schedule.
4. An email is sent to auditee indicating audit schedule.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to reject schedule proposal:



Additional Information*

Open Issues*

2.3.5.5 Use Case «Define new schedule»

Description	If the auditor rejects the auditee proposal for the audit schedule, he has to define a new audit schedule that will not be negotiated.
Scope	Schedule negotiating.
Actor(s)	Auditor and administrator.
Goal	Indicate definitive audit schedule.
Trigger	Rejection of auditee audit schedule proposal.
Frequency*	Depends on the audits frequency but it's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. A first audit schedule proposal was rejected by the auditee.
3. Auditee suggests a new audit schedule.
4. Auditor rejects auditee suggestion.

Minimal Postconditions

Success Postconditions

1. Audit date should be greater than actual date.
2. An email should be sent to the auditee indicating definitive audit schedule.

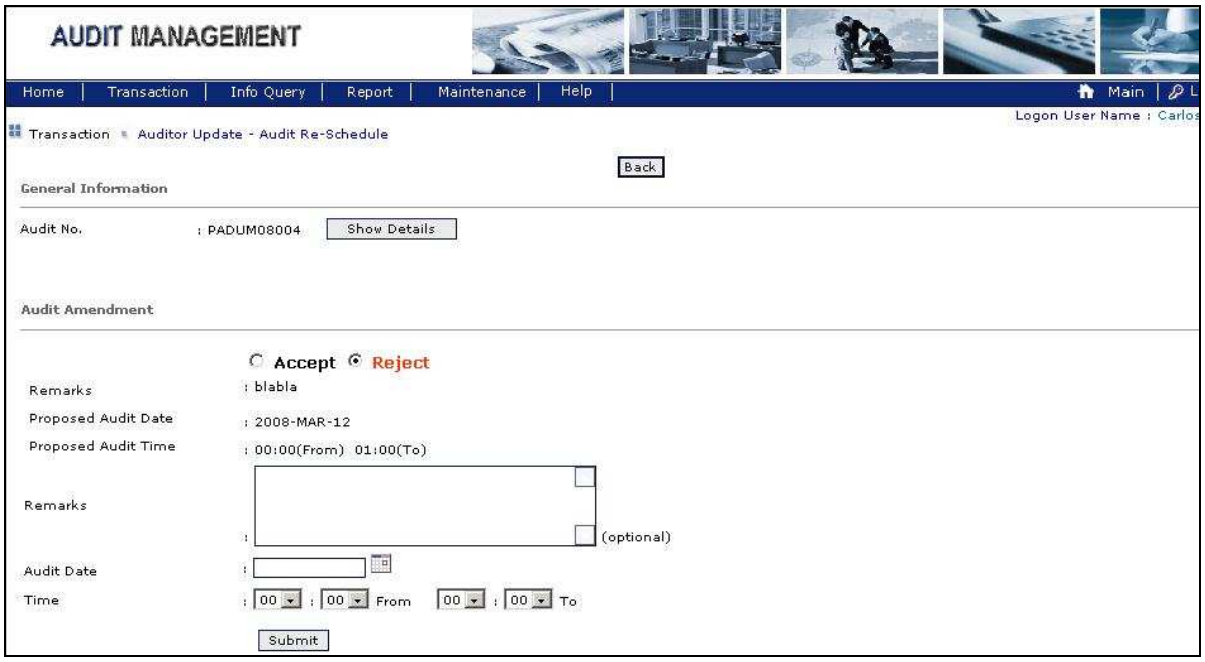
Main Flow

1. Auditor accesses audit page.
2. Auditor rejects auditee proposal for audit schedule.
3. A form to introduce new audit schedule is displayed.
4. Auditor introduce new audit schedule.
5. An email is sent to auditee indicating audit schedule.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

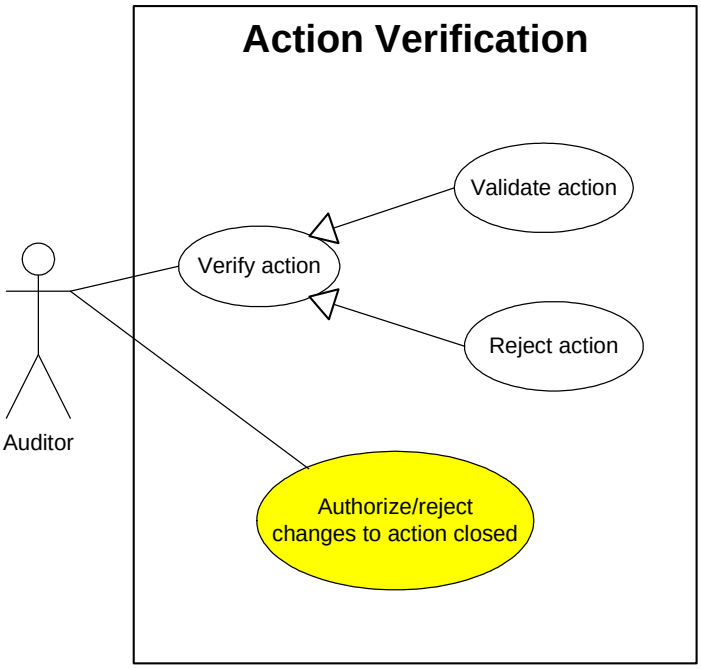
1. User interface to define the audit schedule:



Additional Information*

Open Issues*

2.3.6 Actions verification package



2.3.6.1 Use Case «Verify action»

Description	As consequence of an audit, there are some actions that should be performed to correct the detected deviations. After the action completion, the auditor has to verify if the action was properly done and if it corrects the deviation. Auditor's opinion about the action should be introduced in the Audit Management application.
Scope	Actions verification.
Actor(s)	Auditor and administrator.
Goal	Indicate if action corrected the deviation.
Trigger	Auditee indicates that all actions are completed.
Frequency*	Depends on the audits frequency but it's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. Auditee declares actions as "done".

Minimal Postconditions

Success Postconditions

1. An email should be sent to the auditee indicating the result of the audit verification.

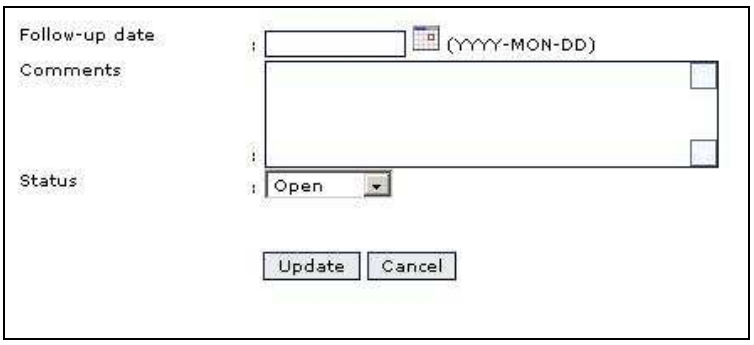
Main Flow

1. Auditor accesses audit page.
2. Auditor edits actions details.
3. Auditor updates action status.
4. An email is sent to auditee indicating action status.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to define action status:



Additional Information*

Actions may have 3 different statuses:

- **Open:** if action isn't completed and due date not overdue.
- **Done:** if action is completed.
- **Not done:** if action isn't finished and due date is overdue.

Open Issues*

1. Auditor may select "Not done" status if he thinks that action don't correct the deviation.

2.3.6.2 Use Case «Validate action»

Description	After verifying the actions results, auditor may validate or reject the action. If the action is done before the due date and contributes to correct the deviation, auditor should validate the action. To validate the action, the auditor has to update the action status to "Done".
Scope	Actions verification.
Actor(s)	Auditor and administrator.
Goal	Indicate that action is successfully done.
Trigger	Auditee indicates that all the actions are completed and auditor starts verifying the actions.
Frequency*	Depends on the audits frequency but it's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. Auditee declares actions as “done”.

Minimal Postconditions

Success Postconditions

1. Action due date isn’t overdue.

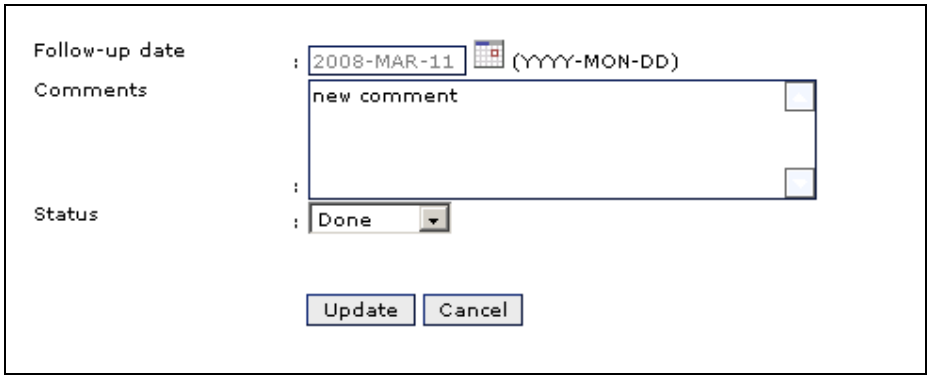
Main Flow

1. Auditor accesses audit page.
2. Auditor edits actions details.
3. Auditor updates action status to “Done”.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

2. User interface to validate action:



Additional Information*

- It is important to do a distinction between the auditee validation and the auditor validation. Auditee puts the action status as “Done” after the action owner perform the action and, after that, the auditor will verify the action and make his own validation, that is, he will put the action status as “Done” or “Not done”. Thus, one action has an auditee status and an auditor status.

Open Issues*

2.3.6.3 Use Case «Reject action»

Description	After verifying the action results, the auditor can validate or reject the action. If the action due date is overdue or the action doesn't contribute to correct the deviation, the auditor has to reject the action. To reject the action, the auditor has to update the action status to "Not Done".
Scope	Actions verification.
Actor(s)	Auditor and administrator.
Goal	Indicate that the action is not successfully done.
Trigger	Auditee indicates that all actions are completed and auditor starts to verify actions.
Frequency*	Depends on the audits frequency but it's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. Auditee declares actions as "done".

Minimal Postconditions

Success Postconditions

1. New action should be created in the Audit Management application with new due date and status "Open".

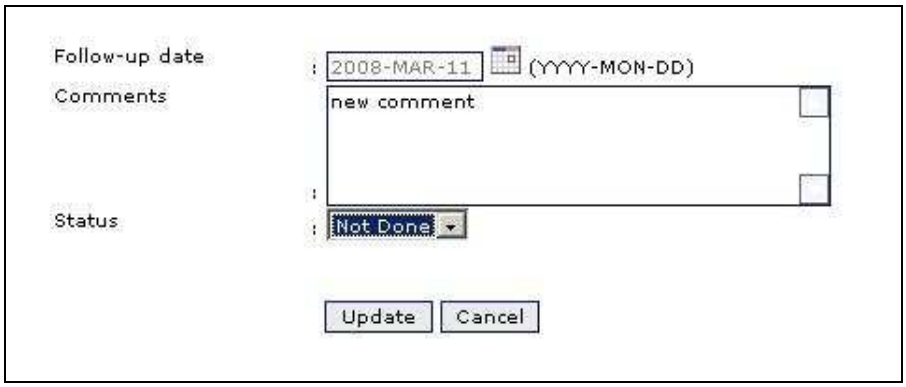
Main Flow

1. Auditor accesses audit page.
2. Auditor edits actions details.
3. Auditor updates action status to "Not Done".

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to reject action:



Additional Information*

- It is important to do a distinction between the auditee validation and the auditor validation. Auditee puts the action status as “Done” after the action owner perform the action and, after that, the auditor will verify the action and make his own validation, that is, he will put the action status as “Done” or “Not done”. Thus, one action has an auditee status and an auditor status.

Open Issues*

1. If the auditor thinks that another kind of action is need to correct the deviation he may insert a comment indicating this.

2.3.6.4 Use Case «Authorize/reject changes to action closed»

Description	After the verification by the auditor, the action can be closed what implies that it can’t be edited. This use case aims to adding the possibility to edit action details even if the action is closed. The changes are performed by the auditee that has to request an authorization from the auditor which can authorize or not the changes.
Scope	Actions verification.
Actor(s)	Auditor and administrator.
Goal	Give or reject auditee permission to edit action closed.
Trigger	Auditee request to do changes to closed action.
Frequency*	It’s predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. Action is closed.
3. Auditee requested an action change.

Minimal Postconditions

Success Postconditions

1. An email should be sent to the auditee with the auditor's decision.

Main Flow

1. Auditee has request an action change.
2. Auditor is notified about the auditee request.
3. Auditor accepts auditee request and defines a due date to auditee achieve changes.
4. An email is sent to auditee with this information.

Alternative Flows / Exceptions*

1. Auditee requested an action change.
2. Auditor is notified about the auditee's request.
3. Auditor rejects the auditee's request and can, optionally, insert a comment to justify his decision.
4. An email is sent to the auditee with this information.

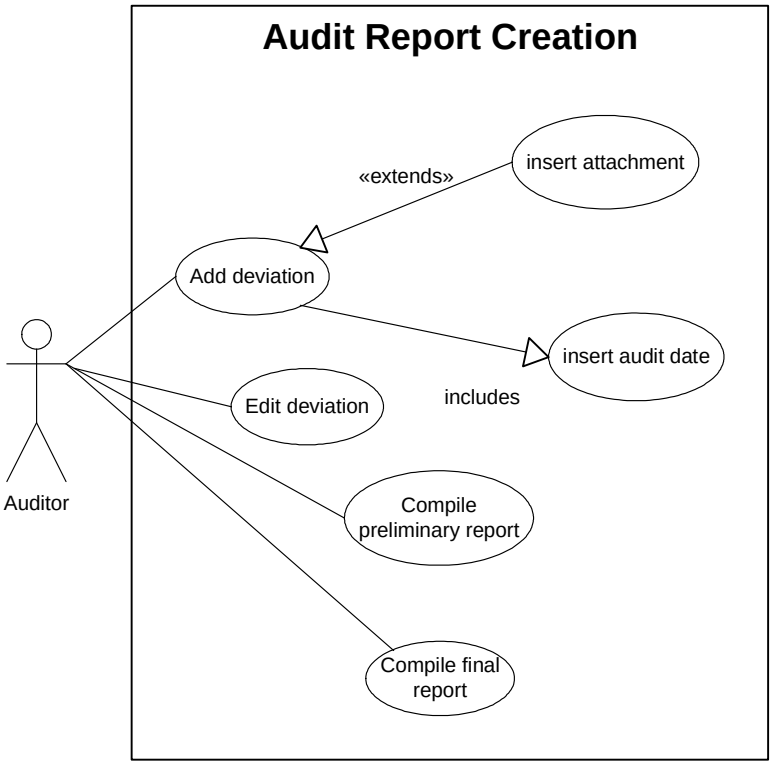
Additional Requirements (non-functional, interface, user interface, etc.)*

Additional Information*

This use case will not be implemented but it is an improvement suggestion which may be discussed with the end users of the application.

Open Issues*

2.3.7 Audit report creation package



2.3.7.1 Use Case «Add deviation»

Description	When the auditor is performing an audit, he encounters some deviations that have to be corrected to maintain the quality patterns. Those deviations have to be added to the audit page in the Audit Management application. This use case includes de insertion of the date when the audit was performed and supports the insertion of an attachment to the audit page.
Scope	Audit report creation
Actor(s)	Auditor and administrator.
Goal	Insert a deviation detected by the auditor while he performs the audit.
Trigger	User action.
Frequency*	Depends on audits' frequency but it's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. Audit schedule has to be defined.
3. Actual date must be greater than audit start date.
4. Preliminary report may not be compiled.

Minimal Postconditions

Success Postconditions

1. The added deviation should stay associated with the audit and has to appear in the audit page.

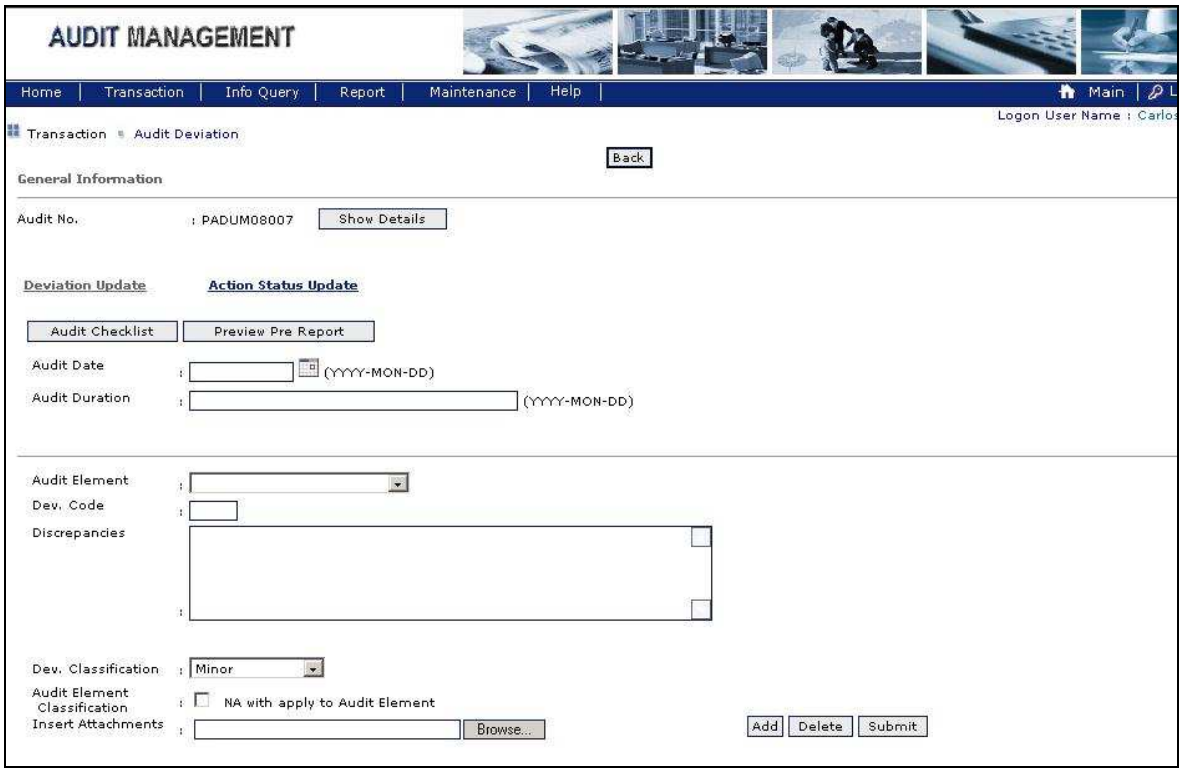
Main Flow

1. Auditor accesses the audit page.
2. Auditor adds deviation details.
3. Auditor adds deviation to the audit.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to add deviation:



Additional Information*

The deviations don't have to be added all at the same time. The auditor may add deviations separately until the compilation of the preliminary report.

Open Issues*

2.3.7.2 Use Case «Compile preliminary report»

Description	After the insertion of all deviations detected by the auditor, he has to indicate that all the deviations have been added. This task is identified as the preliminary report compilation.
Scope	Audit report creation
Actor(s)	Auditor and administrator.
Goal	Compile preliminary audit report and send it to auditee.
Trigger	User action.
Frequency*	Depends on audits frequency but it's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. Auditor has added all the deviations to the audit.
3. The audit shall have, at least, one deviation.

Minimal Postconditions

1. If the preliminary report can't be compiled, all the deviations have to maintain associated to the audit and an error message must be displayed.

Success Postconditions

1. Preliminary report should be sent to auditee.

Main Flow

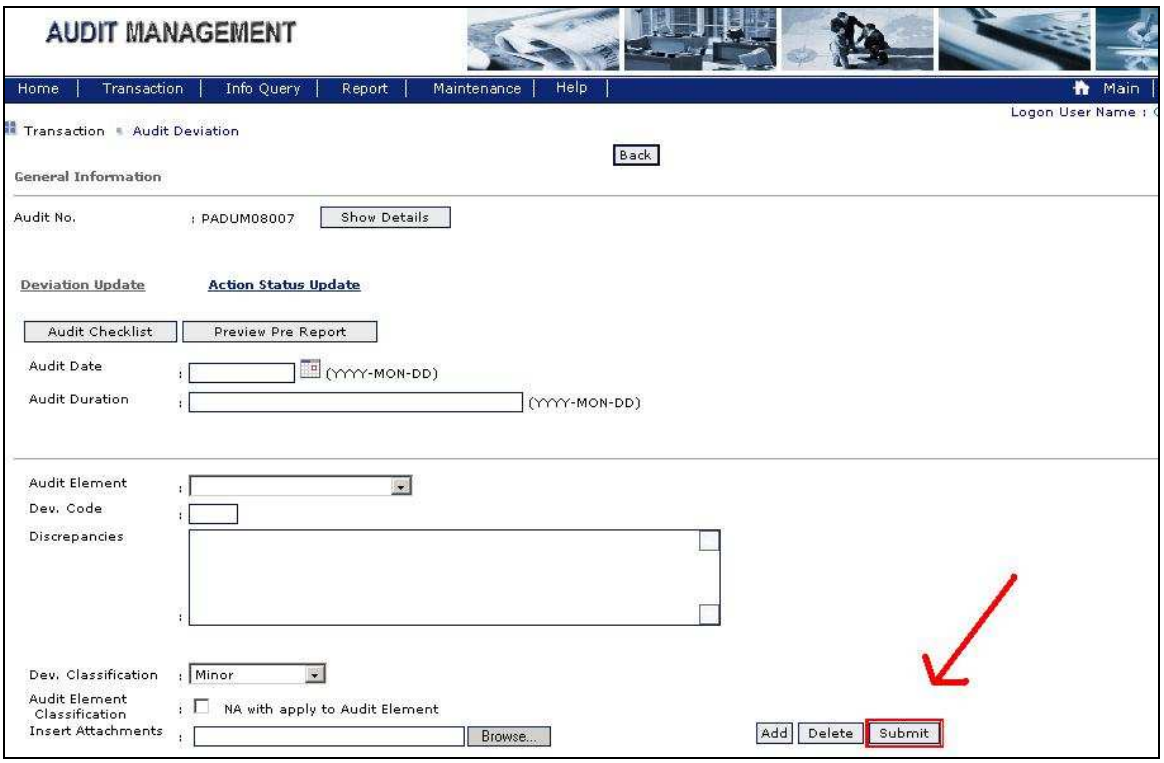
1. Auditor accesses audit page.
2. Auditor submits deviations.

Alternative Flows / Exceptions*

1. In general or external audits there is no discussion with auditees, so, there's no preliminary report. The report compiled is automatically the final report.

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to compile preliminary report:



Additional Information*

The audit report is no more than a collection of deviations. The compilation of the audit report consists on saying that there are no more deviations and the audit report may be generated.

Open Issues*

1. What must happen if the auditor doesn't detect any deviation in the audit?

2.3.7.3 Use Case «Compile final report»

Description	The preliminary audit report may be discussed with the auditee that may disagree with some points. In this case, the auditor should review the audit report and compile a new version that will be the final version. If the auditee doesn't express his disagree with the preliminary report, it will be automatically converted to final report.
Scope	Audit report creation
Actor(s)	Auditor and administrator.
Goal	Compile final audit report and send it to auditee.
Trigger	User action.
Frequency*	Depends on audits frequency but It's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. Preliminary audit report is already compiled.

3. Auditee has accepted preliminary report or auditee response date is overdue.

Minimal Postconditions

1. If final report can't be compiled, all the deviations must maintain associated to the audit and an error message must be displayed.

Success Postconditions

1. Final report should be sent to auditee.

Main Flow

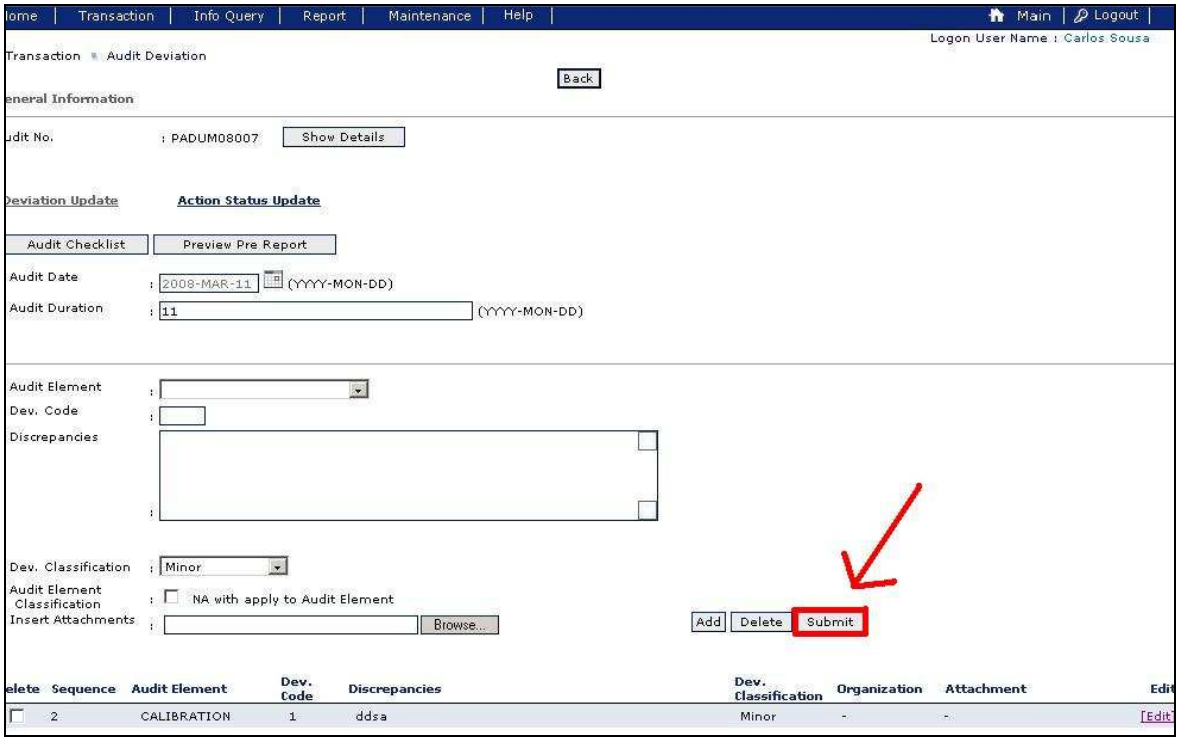
1. Auditor accesses audit page.
2. Auditor edits deviations if necessary.
3. Auditor compiles final report by submission of new deviations.

Alternative Flows / Exceptions*

1. In general or external audits there is no discussion with auditees, so, there's no preliminary report. The report compiled is automatically the final report.
2. If the auditee doesn't respond to the preliminary report before the due date, the report will be automatically converted to final report.

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to submit final report:



Additional Information*

Open Issues*

2.3.7.4 Use Case «Edit deviation»

Description	The deviation attributes may be updated by auditor until the final report compilation.
Scope	Audit report creation
Actor(s)	Auditor and administrator.
Goal	Update deviation details.
Trigger	User action.
Frequency*	Depends on audits frequency but It's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. Final audit report isn't yet compiled.

Minimal Postconditions

Success Postconditions

1. New deviation details should be saved and accessible by the audit page.

Main Flow

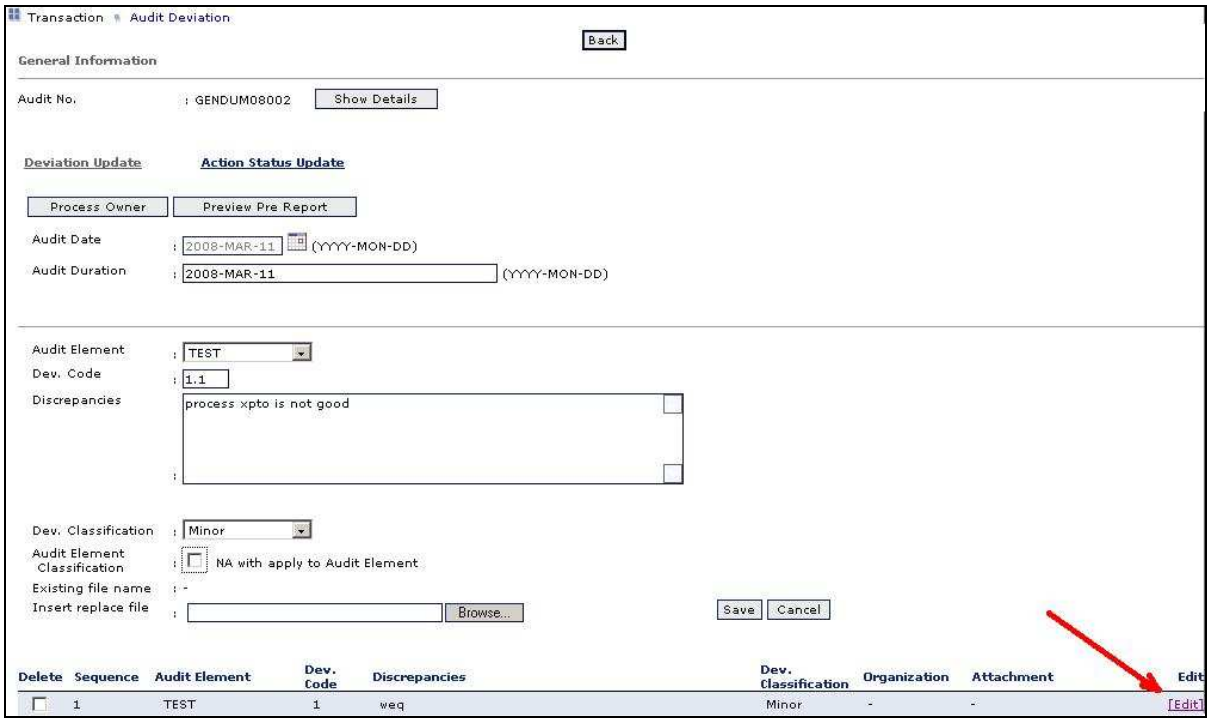
1. Auditor accesses audit page.
2. Auditor accesses deviation details.
3. Auditor edits deviations details.
4. Auditor saves changes done.

Alternative Flows / Exceptions*

1. Auditor may cancel deviation edition and, in this case, deviation will maintain its old details.

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to edit the deviation:

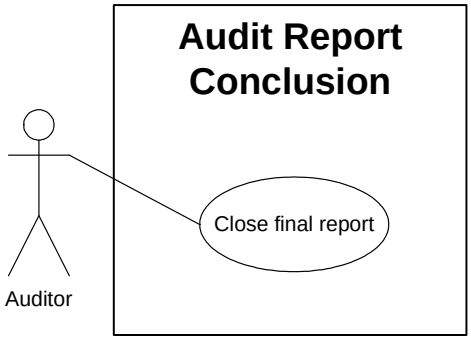


After clicking in the [Edit] option, deviation details will be displayed and can be changed.

Additional Information*

Open Issues*

2.3.8 Audit report conclusion package



2.3.8.1 Use Case «Close final report»

Description After the compilation of the final report, there is a phase where the action owner performs actions necessary to correct the deviations. When all the actions are done, the auditor will verify those actions and, if he thinks that deviations are resolved, he may close the final report. After that, no changes can be made to the audit report.

Scope Audit report conclusion.

Actor(s)	Auditor and administrator.
Goal	Close audit report and set audit as finished.
Trigger	User action.
Frequency*	Depends on audits frequency but It's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditor permissions.
2. All actions statuses are "Done".

Minimal Postconditions

Success Postconditions

1. No more changes may be done to the audit report.
2. Audit report should be sent to the auditee.

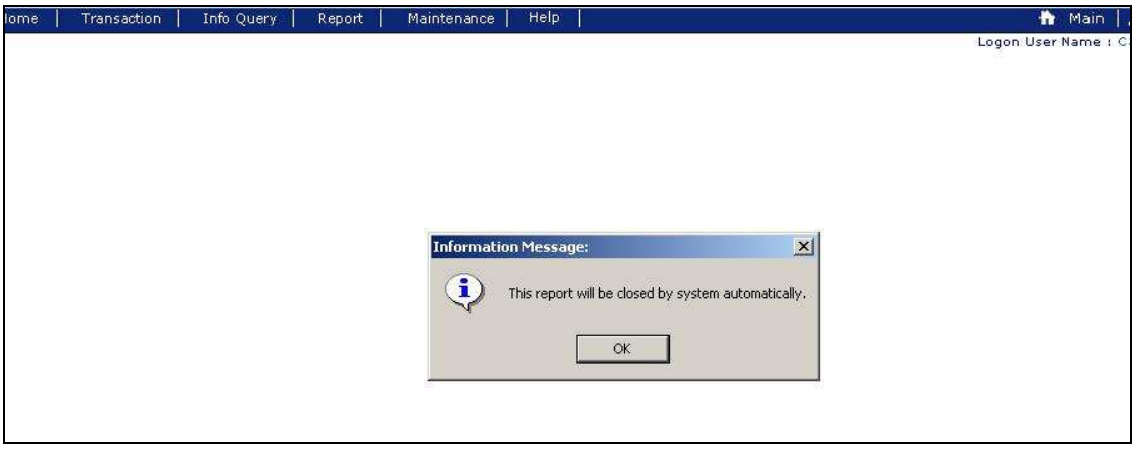
Main Flow

1. Auditor accesses audit page.
2. Auditor accesses action details.
3. Auditor puts all actions statutes "Done".
4. Audit report is automatically closed.
5. Audit report is sent to auditee.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface that appears after the closure of all the actions:



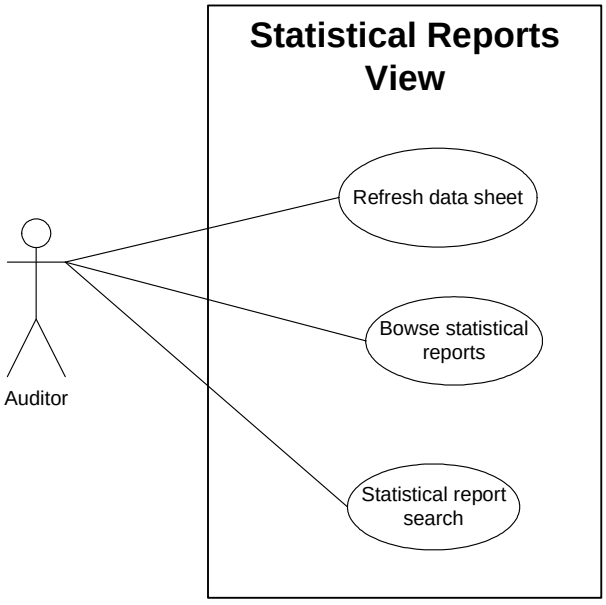
Additional Information*

The auditor closes the final report when he puts status of last action as "Done".

Open Issues*

1. Authorize changes to closed reports.

2.3.9 Statistical reports view package



2.3.9.1 Use Case «Refresh data sheet»

Description	Users may refresh the data sheets present in the MS Excel files for report types. They may refresh all the data sheets or refresh sheets by category.
Scope	Statistics reports
Actor(s)	Auditor and administrator.
Goal	Get latest updated data.
Trigger	User action.
Frequency*	Depends on audits frequency but It's predictable that it will be regularly used.

Preconditions

1. User is logged in the system and has auditor permissions.

Minimal Postconditions

1. A message should be displayed indicating if the operation succeeded or not.

Success Postconditions

1. Data sheet should be immediately available and updated.
2. A message indicating the operation success should be displayed.

Main Flow

1. Auditor accesses audit page.

1. Auditor accesses refresh data sheet page.
2. Auditor selects report type he wants to refresh.
3. Auditor choose to refresh all audit categories or only a specific one.
4. Auditor refresh data sheet and a message is displayed.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to refresh data sheet:

Additional Information*

Open Issues*

1. Auditee may perform these operations?

2.3.9.2 Use Case «Browse statistics report»

Description	There are several statistical reports that can be generated, so, the user can customize the report that he wants to view.
Scope	Statistics reports
Actor(s)	Auditor and administrator.
Goal	Consult some statistic report.
Trigger	User action.
Frequency*	Depends on audits frequency but It's predictable that it will be regularly used.

Preconditions

1. User is logged in the system and has auditor permissions.

Minimal Postconditions

Success Postconditions

1. A report should be generated and downloadable in MS Excel file.

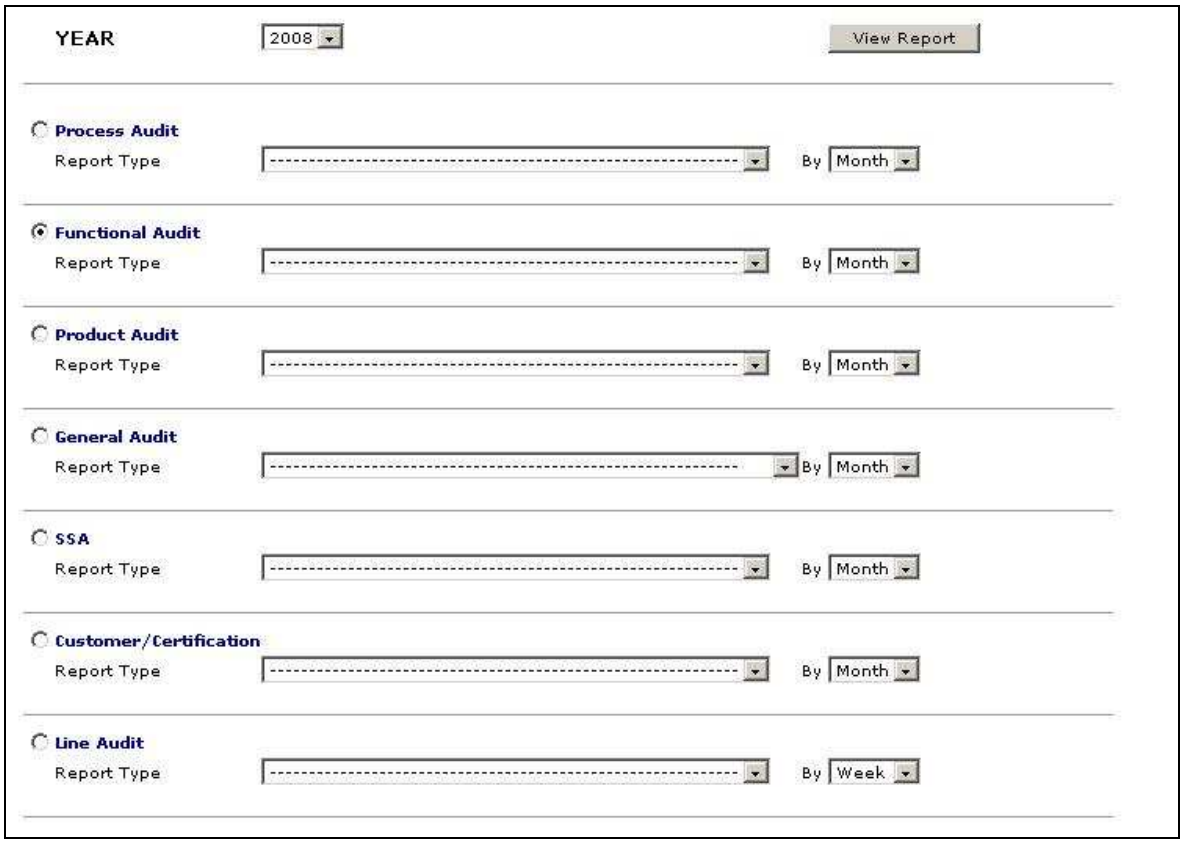
Main Flow

1. Auditor accesses page to browse report.
2. Auditor selects year corresponding to report he wants.
3. Auditor selects report type.
4. Auditor submits information and report is generated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to browse statistics reports:



Additional Information*

Open Issues*

1. Auditee may perform these operations?

2.3.9.3 Use Case «Statistics report search»

Description	User may search statistics report indicating the report type and other details about the report like audit category, action owner or report status.
Scope	Statistics reports
Actor(s)	Auditor and administrator.
Goal	Search statistics reports.
Trigger	User action.
Frequency*	Depends on audits frequency but It's predictable that it will be regularly used.

Preconditions

1. User is logged in the system and has auditor permissions.

Minimal Postconditions

Success Postconditions

1. List of reports that match the query is displayed.

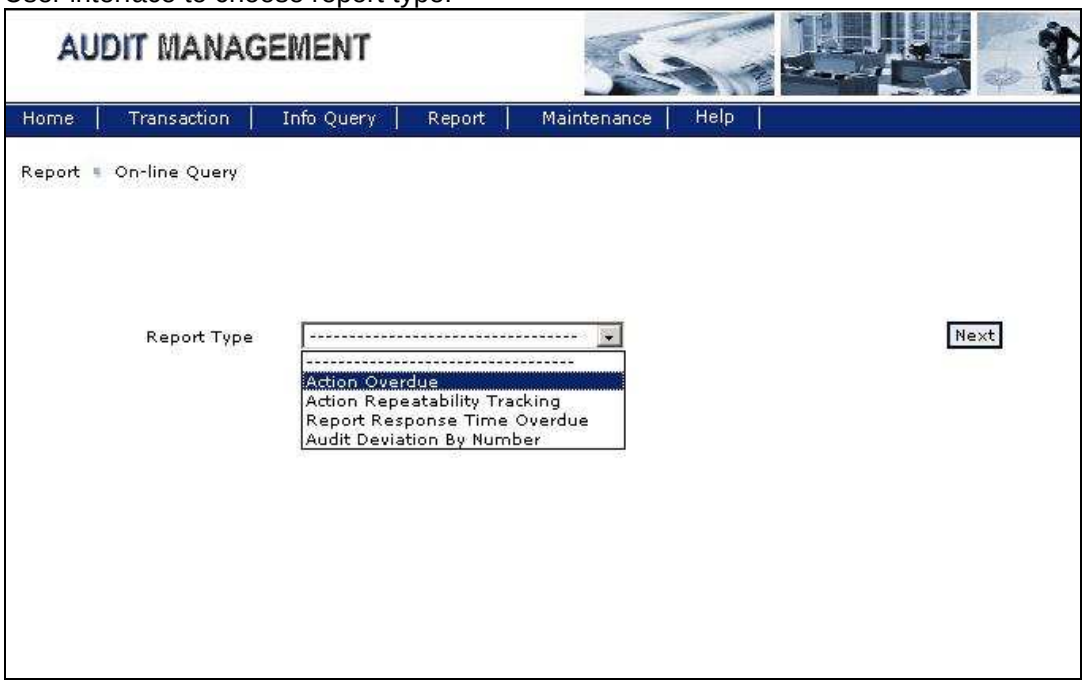
Main Flow

1. Auditor accesses report search page
2. Auditor defines report type.
3. Auditor arrives to report details page.
4. Auditor introduces report details and performs the query.
5. A list with all the reports that match the query is displayed.

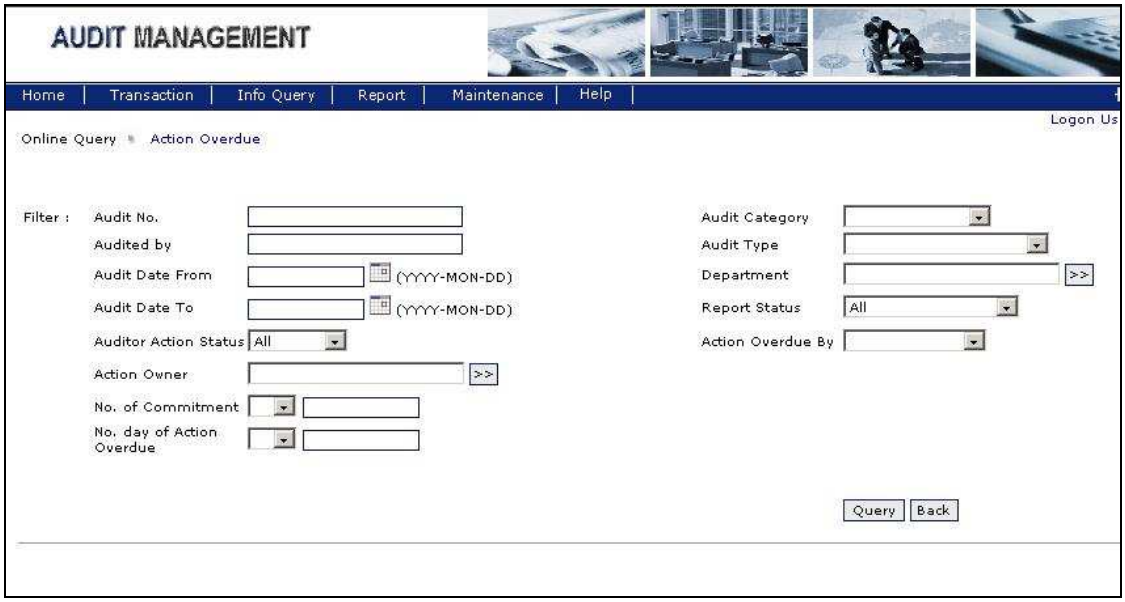
Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to choose report type:



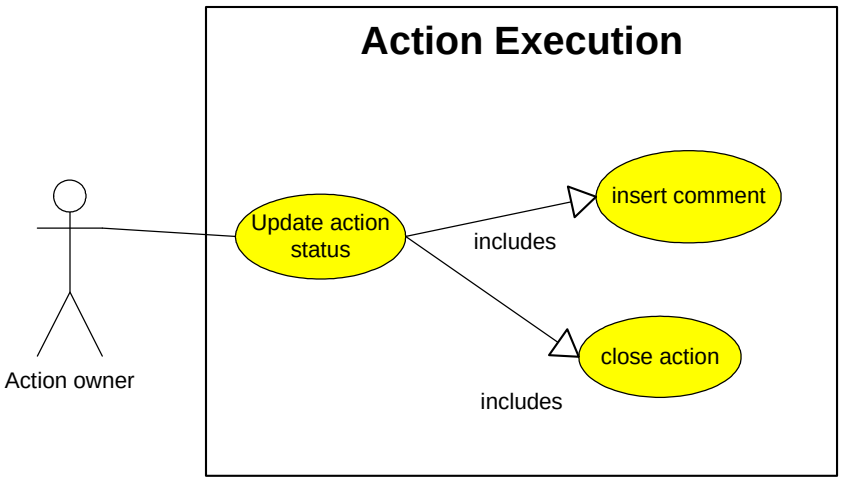
2. User interface to introduce report details:



Additional Information*

Open Issues*

2.3.10 Action achievement package



2.3.10.1 Use Case «Update action status»

Description	Currently, the action owner doesn't interact with the Audit Management application but, with this use case, it's intended to give him a role in the application. After achieving the action, the Action owner should update the status of actions that he's responsible through the insertion of a comment and puts the action as "Done".
Scope	Action achievement.
Actor(s)	Action owner and Administrator.
Goal	Update action status by action owner
Trigger	User action.
Frequency*	Depends on audits frequency but It's predictable that it will be often used.

Preconditions

1. User is logged in the system and has Action owner permissions for this specific action.
2. Action status is "Open".

Minimal Postconditions

Success Postconditions

1. Action status should be as "Done" by Action owner.
2. An email should be sent to auditee indicating that action owner has achieved action.

Main Flow

1. Action owner accesses audit page.
2. Action owner accesses action details.
3. Action owner puts action status as “Done” and may insert a comment.
4. An email is sent to auditee with this information.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

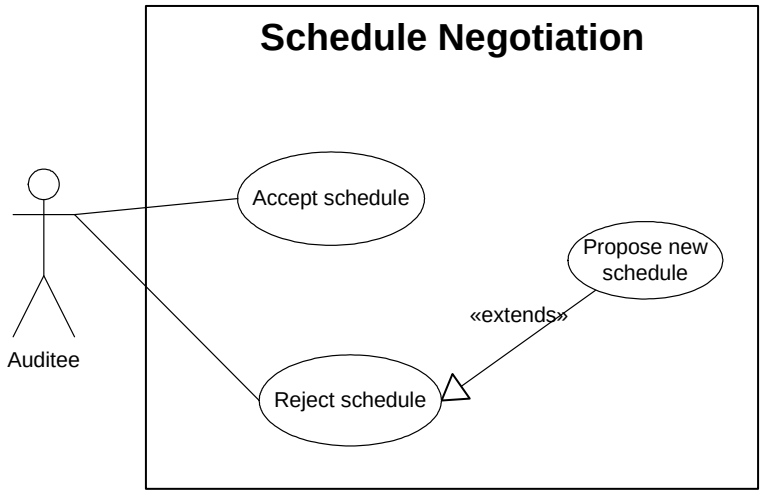
Additional Information*

After the Action owner put status as “Done”, the action has to be validated by the auditee and, then, by the auditor.

Currently, it’s the auditee that performs this update after an offline meeting with the Action owner. The aim of this use case is to put the Action owner interacting with the Audit Management application but, like it is not implemented in actual Audit Management application, it will not be implemented, in a first time, in the new application. This represents an improvement requirement that should be discussed with the application owners and the application users.

Open Issues*

2.3.11 Schedule negotiation package



2.3.11.1 Use Case «Accept schedule»

Description	After the audit schedule proposed by the auditor, the auditee may accept the auditor’s proposal.
Scope	Schedule negotiation.
Actor(s)	Auditee and Administrator.

Goal	Accept audit schedule.
Trigger	Schedule proposal by auditor.
Frequency*	Depends on audits frequency but It's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditee role in this audit.
2. A schedule proposal has already been done.

Minimal Postconditions

Success Postconditions

1. An email should be sent to the auditor indicating that its proposal has been accepted.

Main Flow

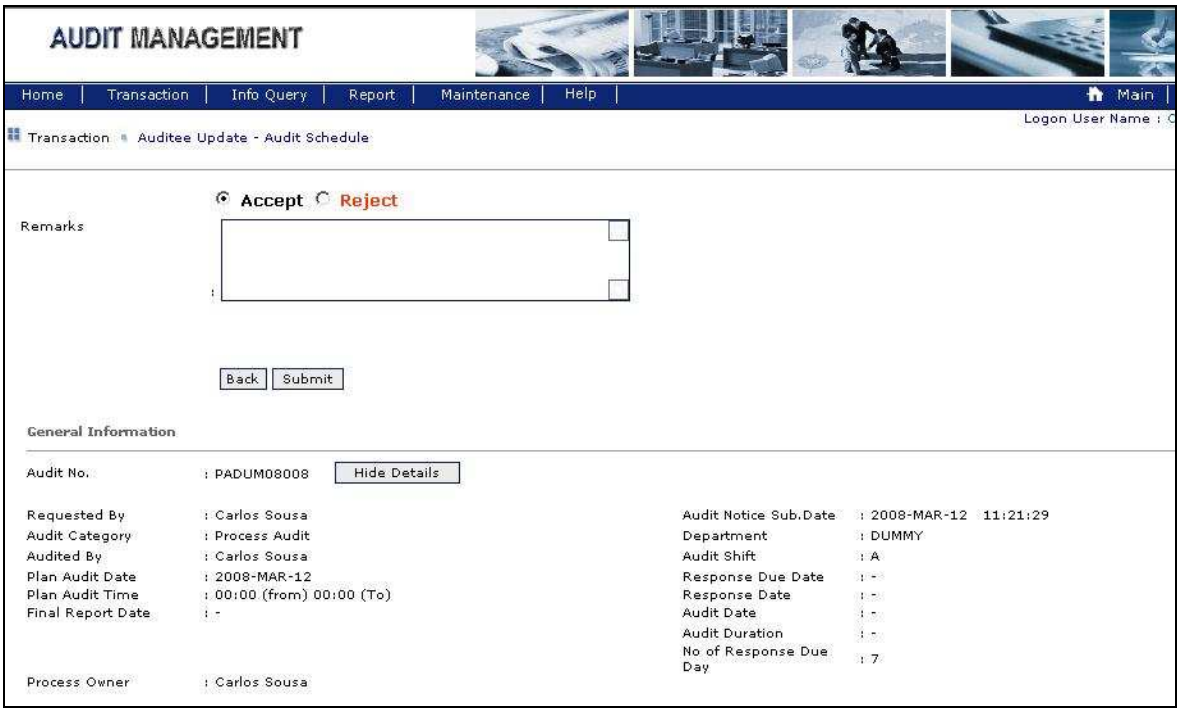
1. Auditee accesses audit page.
2. Audit schedule proposal is displayed.
3. Auditee accept schedule proposal.
4. An email is sent to auditor with auditee response.

Alternative Flows / Exceptions*

1. In general and external audits there's no schedule negotiation.

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to accept the audit schedule:



Additional Information*

Open Issues*

2.3.11.2 Use Case «Reject schedule»

Description	After the audit schedule proposed by the auditor, if the auditee disagrees with the schedule, he may reject the auditor proposal and has to suggest a new schedule. The audit schedule may only be rejected once.
Scope	Schedule negotiation.
Actor(s)	Auditee and Administrator.
Goal	Define another audit schedule.
Trigger	Schedule proposal by auditor.
Frequency*	Depends on audits frequency but It's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has auditee role in this audit.
2. A schedule proposal has already been done.

Minimal Postconditions

Success Postconditions

1. An email should be sent to the auditor with the auditee's response.

Main Flow

1. Auditee accesses audit page.
2. Audit schedule proposal is displayed.

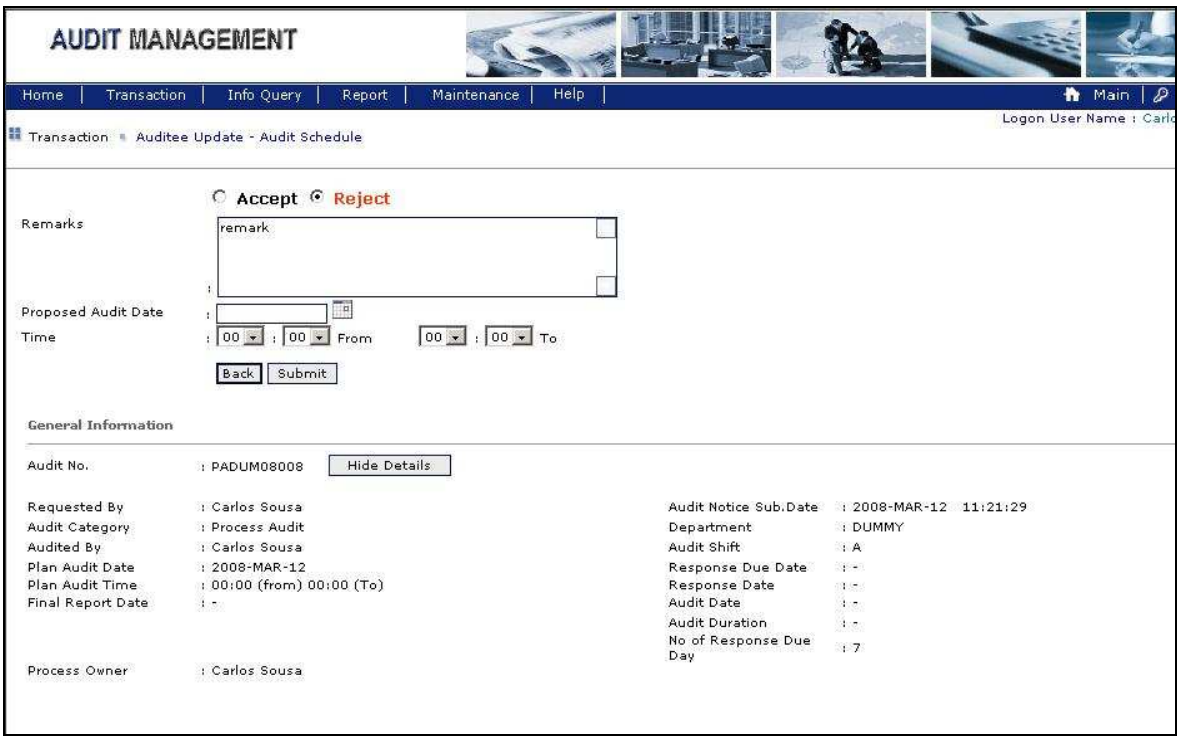
3. Auditee reject schedule proposal and suggests new schedule.
4. An email is sent to auditor with auditee response.

Alternative Flows / Exceptions*

1. In general and external audits there's no schedule negotiation.

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to reject audit schedule:

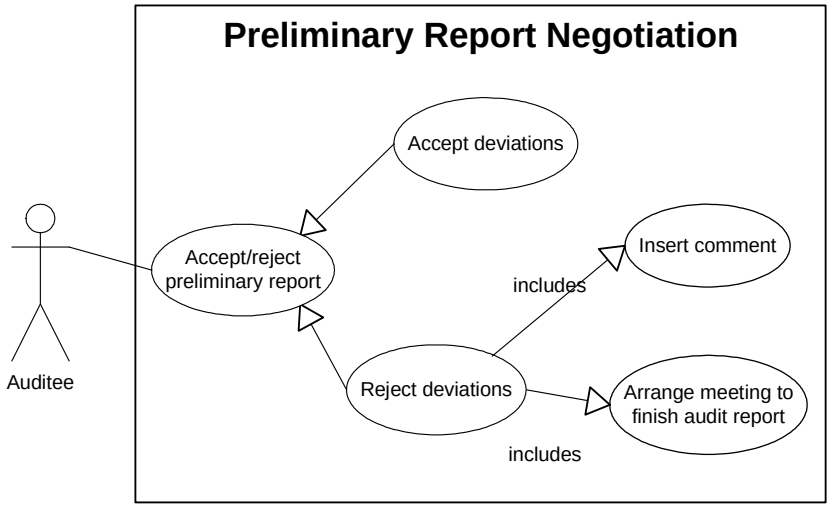


Additional Information*

Open Issues*

1. Auditee proposal for audit schedule may be optional.

2.3.12 Preliminary report negotiation package



2.3.12.1 Use Case «Accept/reject preliminary report»

Description	The auditee can accept or reject the preliminary audit report. This consists on accepting or rejecting all the deviations identified by the auditor present in the preliminary report. If the auditee rejects the preliminary report, he shall insert remarks to justify its rejection and indicate a date for a meeting to discuss the rejected deviations.
Scope	Audit report negotiation.
Actor(s)	Auditee and Administrator.
Goal	Accept or reject preliminary audit report.
Trigger	Preliminary report compilation by auditor.
Frequency*	Depends on audits frequency but It's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditee role in this audit.
2. Preliminary audit report is already compiled.

Minimal Postconditions

Success Postconditions

1. An email should be sent to the auditor with auditee's response.

Main Flow

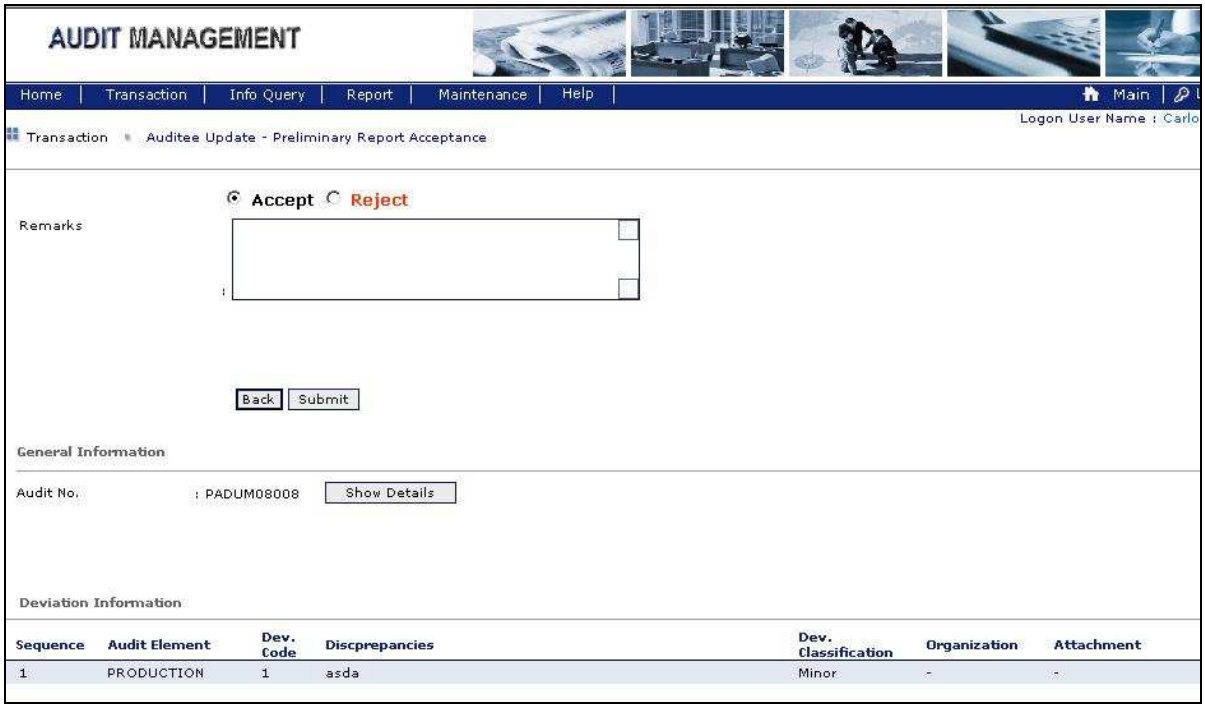
1. Auditee accesses audit page.
2. Deviations that compose audit report are displayed.
3. If auditee agrees with deviations he accepts the report, else he rejects it.
4. An email is sent to auditor with auditee response.

Alternative Flows / Exceptions*

1. In general and external audits there's no discussion about audit report.

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to accept or reject preliminary audit report:

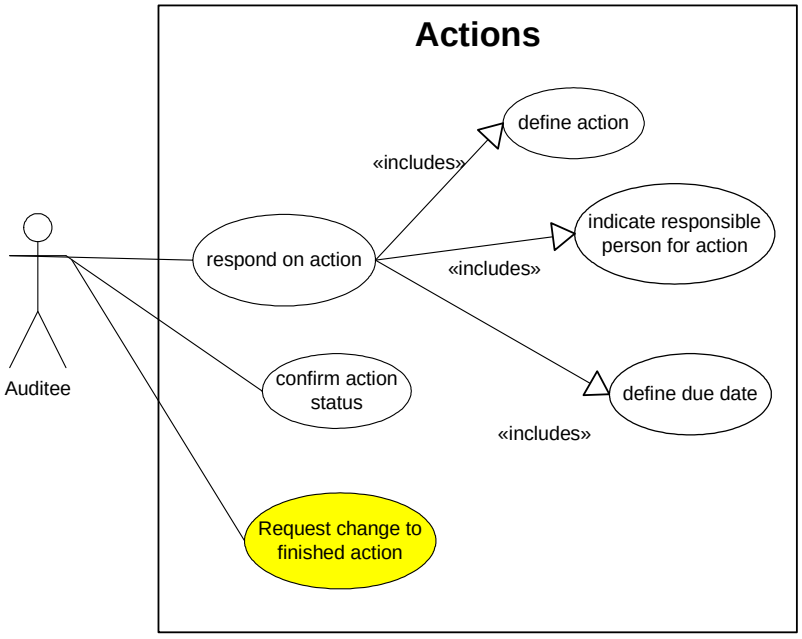


Additional Information*

Open Issues*

1. Currently, the deviations are accepted or rejected like a whole but it should be possible to accept or reject them separately.

2.3.13 Actions package



2.3.13.1 Use Case «Respond on action»

Description	After the compilation of the final audit report, the auditee has to respond with actions to correct the deviations. The auditee has to define action details, indicate the person which will be responsible for performing the action and define the due date for the action.
Scope	Actions definition.
Actor(s)	Auditee and Administrator.
Goal	Define action to contribute to deviation correction.
Trigger	Final audit report compilation.
Frequency*	Depends on audits frequency but It's predictable that it will be often used.

Preconditions

1. User is logged in the system and has auditee role in this audit.
2. Final audit report is already compiled.
3. Audit report still open.

Minimal Postconditions

Success Postconditions

1. Actions details should be immediately available and updated.
2. Action owners should be notified by email about actions they have to perform.

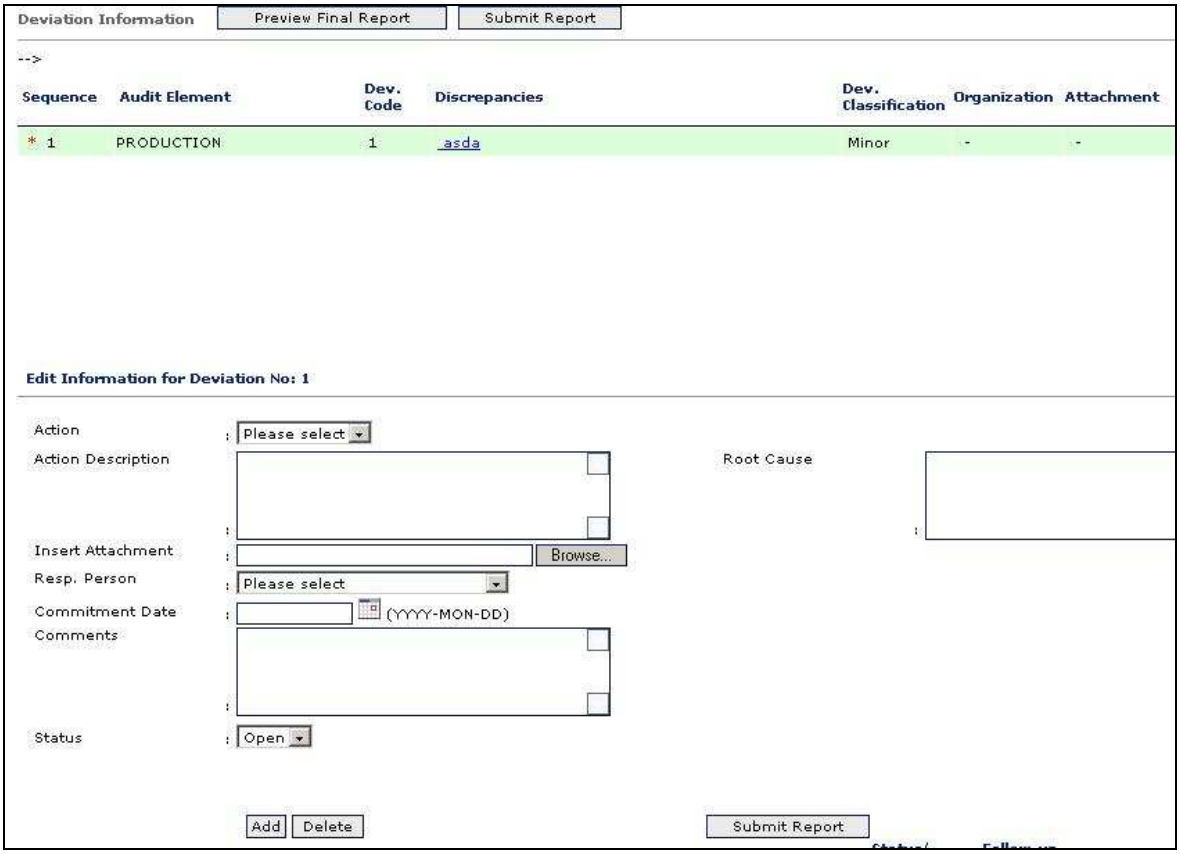
Main Flow

1. Auditee accesses audit page.
2. Auditee access deviations details.
3. Auditee defines action(s) to correct deviations.
4. Auditee assigns responsible person to perform action.
5. Auditee saves action and an email is sent to person responsible for action.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to add actions:



Additional Information*

Every deviation should have, at least, one action.

Open Issues*

2.3.13.2 Use Case «Confirm action status»

Description After action's achievement by the responsible person, the auditee has to confirm that action is done. It is necessary the auditee confirmation to advertise the auditor that the action was performed.

Scope	Actions achievement.
Actor(s)	Auditee and Administrator.
Goal	Confirm that action is achieved.
Trigger	Action achievement by Action owner.
Frequency*	Depends on audits frequency but It's predictable that it will be often used.

Preconditions

1. Action is achieved by the Action owner.

Minimal Postconditions

Success Postconditions

1. Action status should be updated to "Done" by auditee.
2. An email should be sent to the auditor indicating that action is done.

Main Flow

1. Auditee accesses audit page.
2. Auditee accesses action details.
3. Auditee puts status to "Done" and inserts the completion date.
4. Auditee save changes and an email is sent to auditor.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to update action status:

Additional Information*

Open Issues*

2.3.13.3 Use Case «Request change to finished action»

Description	Currently, if the action status is "Done" by the auditor, the auditee can't update action details. This use case aims to give to auditee this possibility but, for that, he needs to have the auditor's authorization which may be

	requested across the Audit Management application.
Scope	Actions definition.
Actor(s)	Auditee and Administrator.
Goal	Update details of action finished.
Trigger	User action.
Frequency*	It's predictable that it will be regularly used.

Preconditions

1. User is logged in the system and has auditee role in this audit.
2. Action is closed.
3. Audit report still open.

Minimal Postconditions

Success Postconditions

1. An email should be sent to the auditor with auditee's request.

Main Flow

1. Auditee accesses audit page.
2. Auditee accesses action details.
3. Auditee request action change.
4. An email is sent to auditor with auditee request.

Alternative Flows / Exceptions*

1. In general and external audits this use case will not apply.

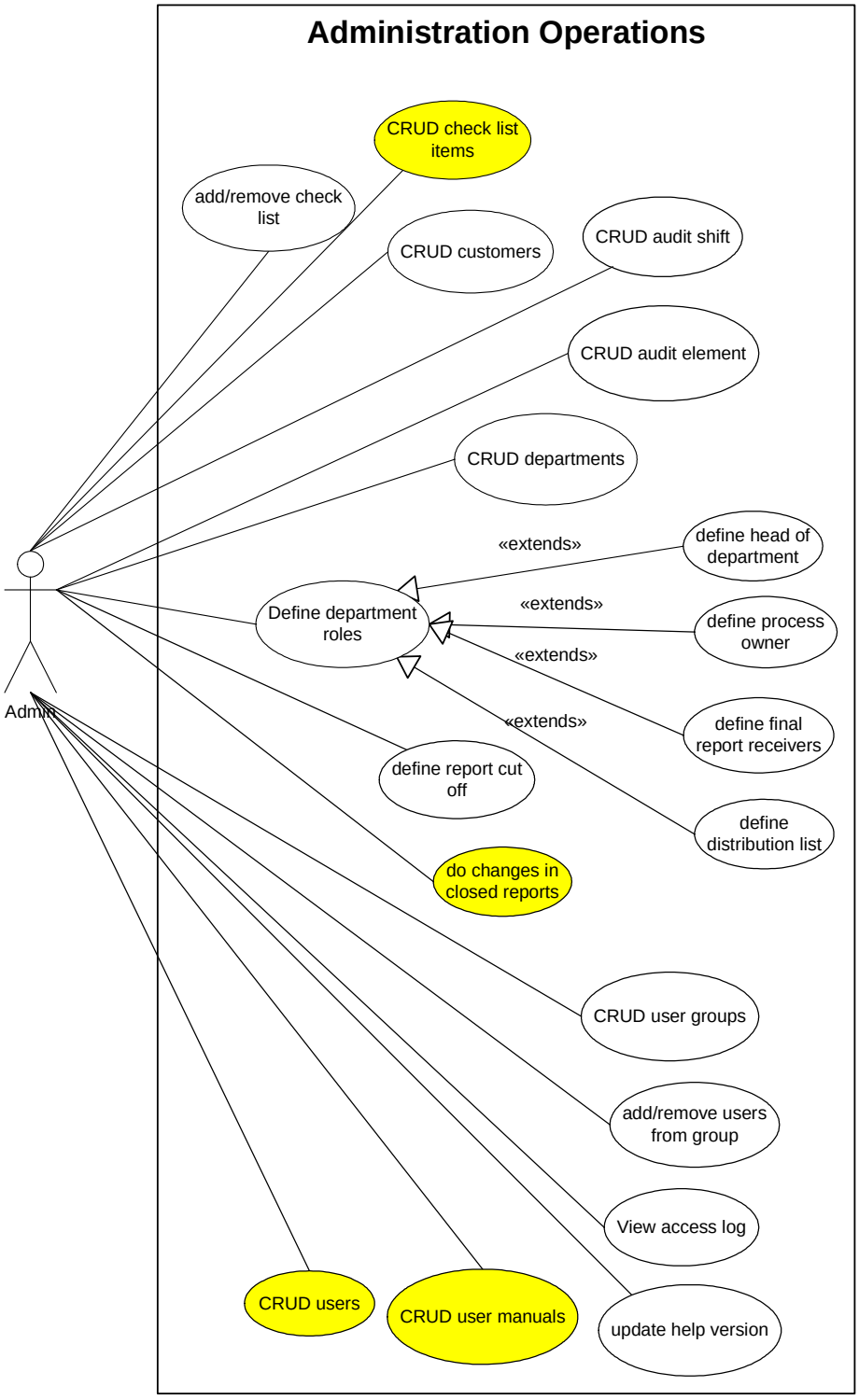
Additional Requirements (non-functional, interface, user interface, etc.)*

Additional Information*

This use case will not be implemented but it is an improvement proposal that shall be discussed.

Open Issues*

2.3.14 Administration operations package



2.3.14.1 Use Case «Add/remove checklist»

Description	Each audit category has a list of items that should be checked. Those checklists may be added or removed from the Audit Management application.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Add or remove checklist associated to an audit category.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has the administrator role.
2. Checklist must be added to an existing audit category.

Minimal Postconditions

1. An error message should be showed if the user try to add more than one audit checklist to the same audit category.
2. An error message should be showed if checklist upload fails.

Success Postconditions

1. An audit category should have, at maximum, one audit checklist.
2. New audit checklist should be available after checklist upload.

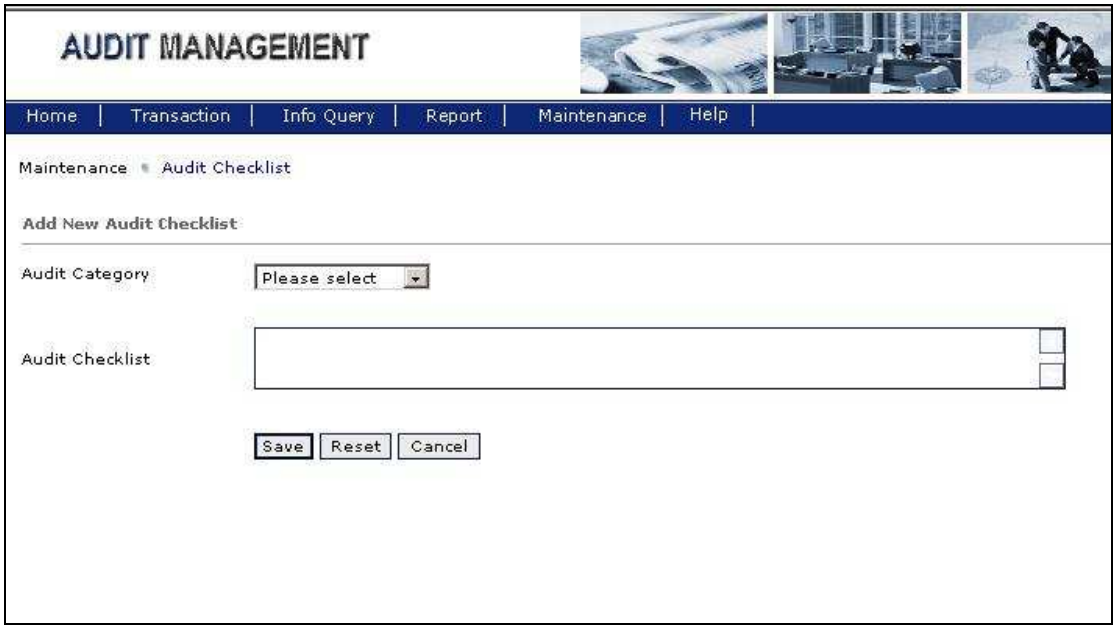
Main Flow

1. Administrator accesses the audit checklist page.
2. Administrator accesses page to add new checklist.
3. Administrator uploads checklist.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to add new audit checklist:



Additional Information*

Currently, those checklists are in MS Excel files.

Open Issues*

1. What are the benefits to have checklists viewable in web browser instead of have to download and open it in MS Excel?

2.3.14.2 Use Case «CRUD checklist items»

Description	CRUD (create, retrieve, update and delete) checklist items across the Audit Management application.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Update checklists across Audit Management application.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has administrator role.
2. Checklist shall be in Audit Management application.

Minimal Postconditions

1. An error message should be showed if user tries to add an item that already exists.
2. An error message should be showed if CRUD operation fails.

Success Postconditions

1. Audit checklist should be immediately available and updated.

Main Flow

1. Administrator accesses the audit checklist page.
2. Administrator does CRUD operation.
3. Audit checklist is automatically updated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. This use case requires that checklist items may be viewable in web browser.

Additional Information*

This use case will not be implemented but is an improvement proposal that shall be discussed.

Open Issues*

2.3.14.3 Use Case «CRUD customers»

Description	CRUD customers of Qimonda that may request external audits.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Update customers in Audit Management database.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has administrator role.

Minimal Postconditions

1. An error message should be showed if user tries to add a customer that already exists.
2. An error message should be showed if CRUD operation fails.

Success Postconditions

1. List of customers should be immediately available and updated.

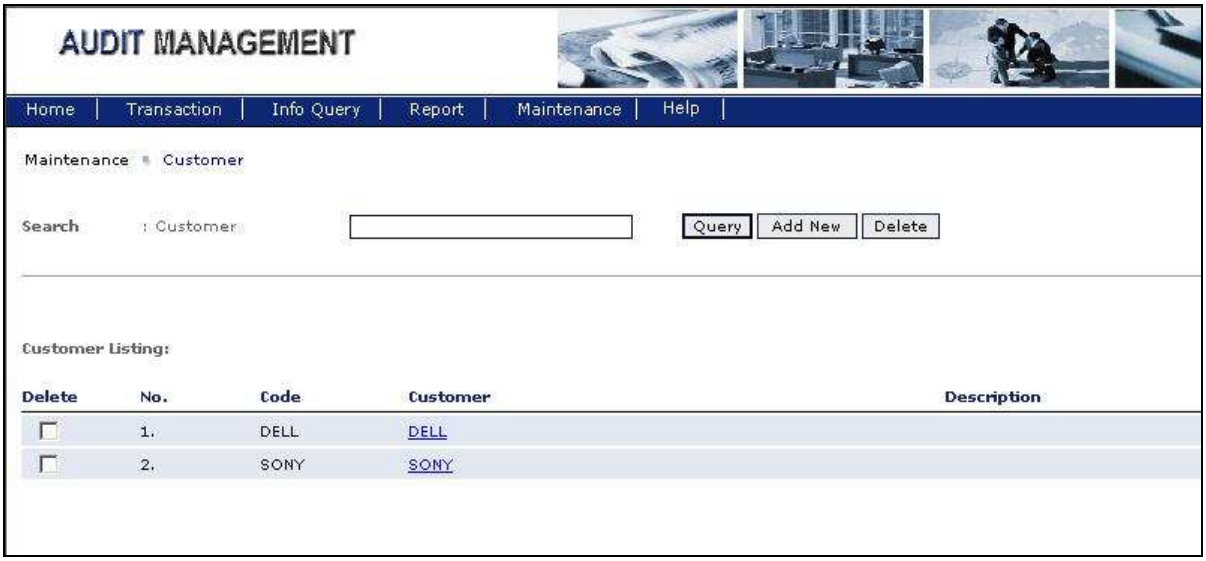
Main Flow

1. Administrator accesses customer's page.
2. Administrator does CRUD operation.
3. List of customers is automatically updated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface of customer's page:



Additional Information*

Open Issues*

2.3.14.4 Use Case «CRUD audit shift»

Description	CRUD shifts when audits may be performed.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Update audit shifts.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has administrator role.

Minimal Postconditions

1. An error message should be showed if user tries to add a shift that already exists.
2. An error message should be showed if CRUD operation fails.

Success Postconditions

1. List of audit shifts should be immediately available and updated.

Main Flow

1. Administrator accesses audit shift's page.
2. Administrator does CRUD operation.
3. List of audit shifts is automatically updated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface of audit shift's page:

Additional Information*

Open Issues*

2.3.14.5 Use Case «CRUD audit elements»

Description	CRUD audit elements associated to audit categories.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Update audit elements.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has administrator role.

Minimal Postconditions

1. An error message should be showed if user tries to add an element that already exists.
2. An error message should be showed if CRUD operation fails.

Success Postconditions

1. List of audit elements should be immediately available and updated.
2. Administrator can't add an audit element that already exists in the audit category

Main Flow

1. Administrator accesses audit elements page.
2. Administrator does CRUD operation.
3. List of audit elements is automatically updated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface of audit elements page:

Additional Information*

Open Issues*

2.3.14.6 Use Case «CRUD departments»

Description	CRUD departments where audits may be performed.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Update departments.
Trigger	User action.
Frequency*	It’s predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has the administrator role.

Minimal Postconditions

1. An error message should be showed if user tries to add a department that already exists.
2. An error message should be showed if CRUD operation fails.

Success Postconditions

1. List of departments should be immediately available and updated.
2. There can’t be two or more departments with the same name.

Main Flow

1. Administrator accesses department’s page.
2. Administrator does CRUD operation.
3. List of departments is automatically updated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface of department’s page:

Additional Information*

Open Issues*

2.3.14.7 Use Case «Define department roles »

Description	This use case allows defining the person that is the head of the department, the process owner, the final report receivers and a distributor list.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Define persons that match the department roles.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has administrator role.
2. Department is already created.
3. Persons to take some role have to be registered in Audit Management database.

Minimal Postconditions

1. An error message should be showed if operation fails.

Success Postconditions

1. Department's roles should be immediately available and updated.

Main Flow

1. Administrator accesses department's roles page.
2. Administrator accesses some department page.
3. Administrator updates persons and roles.
4. Department's roles are automatically updated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface to update some department's roles:

Additional Information*

Open Issues*

2.3.14.8 Use Case «Define report cutoff»

Description	Define date for report cutoff.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Define date for report cutoff.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has the administrator role.

Minimal Postconditions

1. An error message should be showed if report cutoff definition fails.

Success Postconditions

1. New report cutoff date should be immediately available and updated.

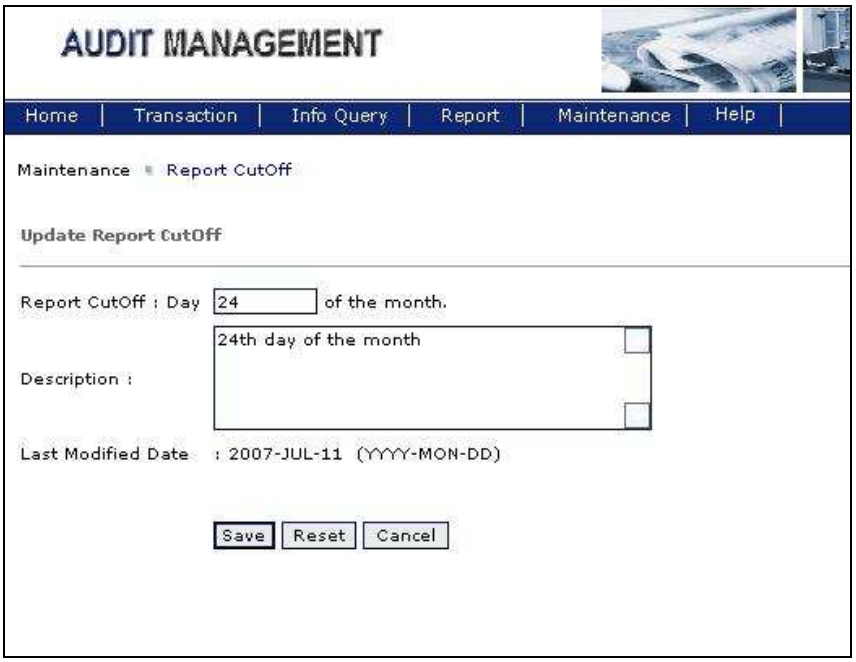
Main Flow

1. Administrator accesses report cutoff page.
2. Administrator updates report cutoff.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface of report cutoff definition page:



Additional Information*

Open Issues*

2.3.14.9 Use Case « Do changes in closed report »

Description	Currently, if a report is closed, there's impossible to do any change in it. This use case aims to give to the administrator the possibility to perform some changes in a closed report.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Update details of a closed report.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has administrator role.
2. Audit report is closed.

Minimal Postconditions

Success Postconditions

1. Changes done in audit report should be immediately available and updated.

Main Flow

1. Administrator accesses audit page.
2. Administrator updates audit details.
3. Administrator closes again audit report.
4. Audit report is sent again to all involved persons.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

Additional Information*

This use case will not be implemented but is an improvement proposal that shall be discussed.

Open Issues*

2.3.14.10 Use Case « CRUD user groups »

Description	A user group is a set of permissions that can be attributed to the Audit Management users. It will be possible to create new groups, update and delete existing groups.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Update user groups.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User is logged in the system and has administrator role.

Minimal Postconditions

1. An error message should be showed if CRUD operation fails.

Success Postconditions

1. Changes done to user groups should be immediately available and updated.

Main Flow

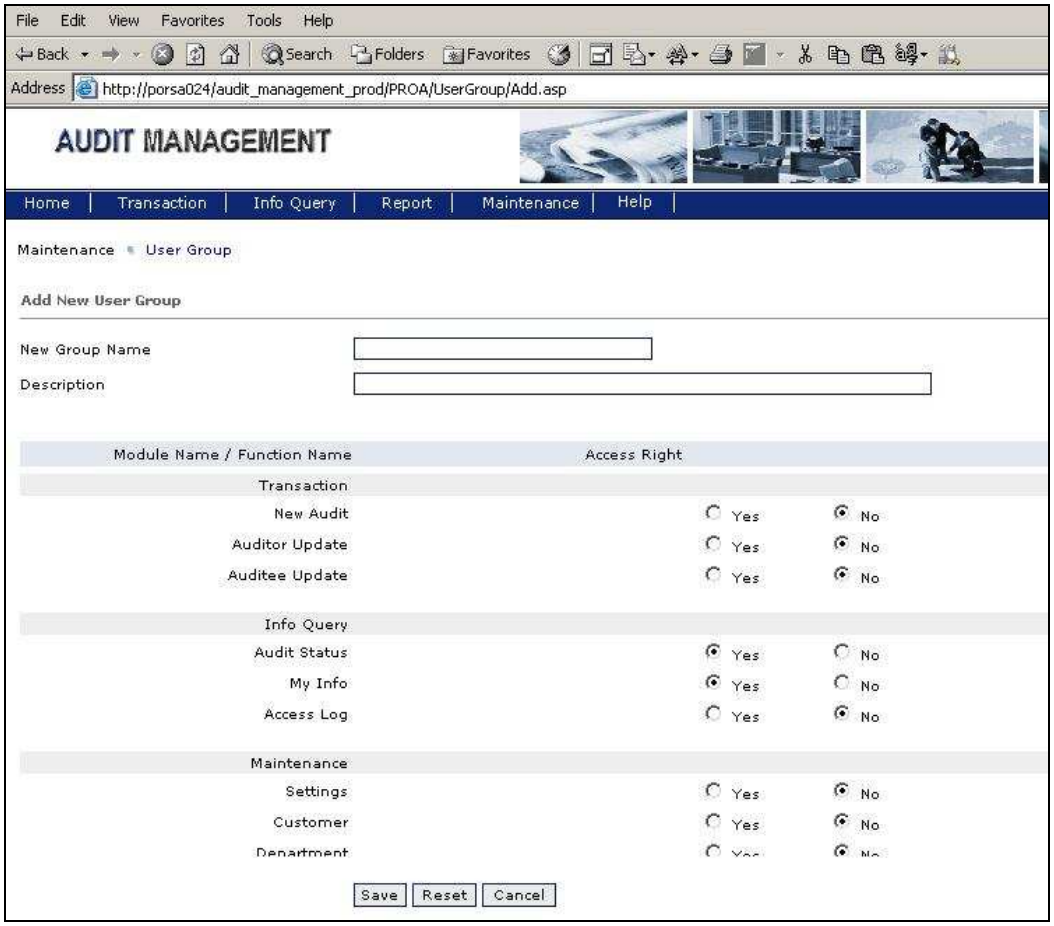
1. Administrator accesses user groups page.
2. Administrator does CRUD operation.
3. List of user groups is automatically updated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

1. User interface of list of user groups:

2. User interface to define permissions of a new user group:



Additional Information*

Open Issues*

2.3.14.11 Use Case «Add/remove users from group»

Description	Users of Audit Management application may belong to several user groups, so, this use case aims to add or remove a user to some group.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Add or remove a user from some user group.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User authorized to do addition or removal operation should be logged in the system and should have administrator role.

Minimal Postconditions

1. An error message should be showed if addition or removal operation fails.

Success Postconditions

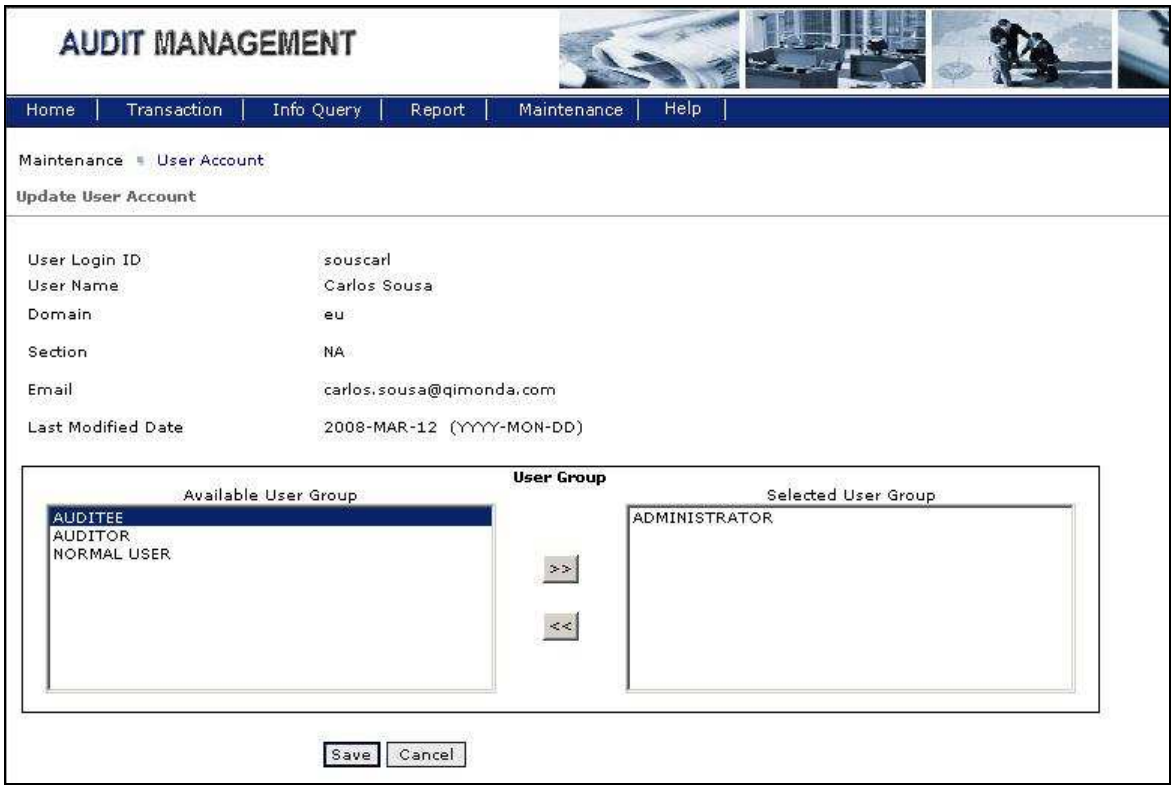
1. Users shall belong, at least, to one user group.
2. Changes done in user groups should be immediately available and updated.

Main Flow

1. Administrator accesses user's page.
2. Administrator selects user that he wants to update permissions.
3. Administrator adds or removes user from groups.
4. Administrator saves changes.
5. User permissions are automatically available and updated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*



Additional Information*

A user may belong, at the same time, to more than one user groups.

Open Issues*

2.3.14.12 Use Case «View access log»

Description	User login and logout details between some dates will be displayed in the access log.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	View login and logout details from users.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User to view access log should be logged in the system and should have administrator role.

Minimal Postconditions

Success Postconditions

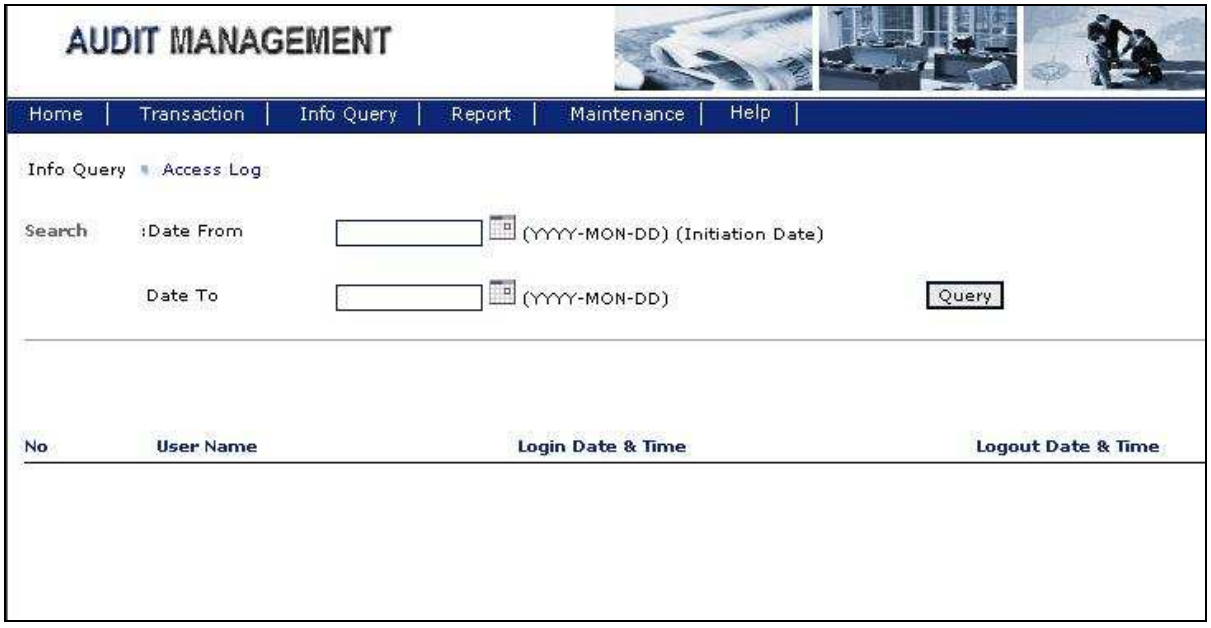
1. A list with user login and logout details between defined dates should be displayed.

Main Flow

1. Administrator accesses access log page.
2. Administrator introduce interval of dates.
3. User login and logout details are displayed.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*



Additional Information*

Open Issues*

2.3.14.13 Use Case «Update help version»

Description	There should be a log of versions of help menu, so, when a change is performed in the help menu it has to be registered.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Update version and details of help menu.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User should be logged in the system and should have administrator role.

Minimal Postconditions

1. An error message should be showed if update operation fails.

Success Postconditions

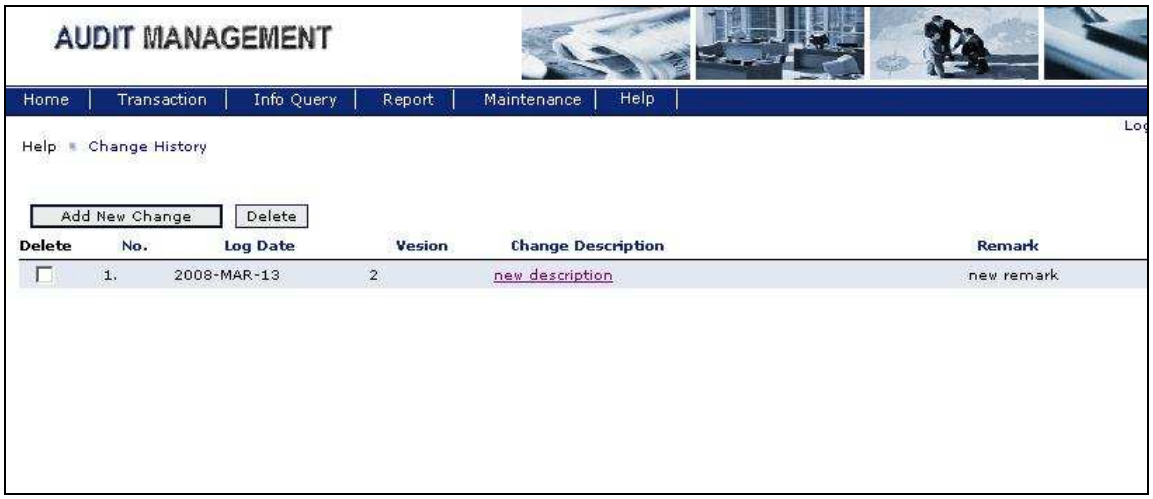
1. Details of new version of help menu should be displayed.

Main Flow

1. Administrator accesses help history page.
2. Administrator adds a description to new help version.
3. Administrator saves changes.
4. Help history is updated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*



Additional Information*

Open Issues*

2.3.14.14 Use Case «CRUD users»

Description	Currently, the users are imported from YODA middleware and are stored in the Audit Management database. This is an increased difficulty to add more users to the Audit Management application. With this use case, it is expected than users can be added or removed directly in the Audit Management application.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Add, remove or update Audit Management users
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. The user should be logged in the system and should have administrator role.

Minimal Postconditions

1. An error message should be showed if CRUD operation fails.

Success Postconditions

1. There should not have more than one user with the same username.
2. New user details should be immediately available and updated.
3. An email should be sent to user involved in CRUD operation.

Main Flow

1. Administrator accesses user's page.
2. Administrator performs CRUD operation.
3. An email is sent to user with new user details.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

Additional Information*

This use case will not be implemented but it represents an improvement suggestion and should be discussed with application owners and users.

Open Issues*

2.3.14.15 Use Case «CRUD user manuals»

Description	Currently, there are three user guides and there's no possibility to add another manual or remove one existent. This use case aims to provide the possibility to perform CRUD operations with user manuals.
Scope	Administration operations.
Actor(s)	Administrator.
Goal	Add, remove or update user manuals.
Trigger	User action.
Frequency*	It's predictable that it will be rarely used.

Preconditions

1. User should be logged in the system and should have administrator role.
2. User manual has to be previously achieved in some format that can be uploaded (.ppt, .pdf, .doc, etc…).

Minimal Postconditions

1. An error message should be showed if CRUD operation fails.

Success Postconditions

1. User manuals should be immediately available and updated.

Main Flow

1. Administrator accesses user manual's page.
2. Administrator performs CRUD operation.
3. User manual's page is updated.

Alternative Flows / Exceptions*

Additional Requirements (non-functional, interface, user interface, etc.)*

Additional Information*

This user case will not be implemented but it represents an improvement suggestion and should be discussed with application owners and users.

Open Issues*

2.4 External interfaces

2.4.1 User interfaces (GUI)

The Audit Management homepage is showed below and contains pending items from logged user. Other user interfaces may be consulted in chapters related to the use cases description.

2.4.2 System interfaces

The Audit Management application needs to interface with YODA middleware to get user details for users that may authenticate in the application. It's also needed to interface with an SMTP server because application has to notify users via email.

2.5 Quality requirements

Quality features are created by the authors of the requirements specification in close cooperation with the QA manager and in coordination with users/clients. This section is used to specify the quality requirements of the product (product features).

Object	Value
Reliability	Defines the length of time permitted before the system must be available once more following a crash
Robustness	Describes system security, e.g. how many system crashes are tolerated in a time frame of, e.g. 3 months

Object	Value
Function support	Completeness: degree to which the required or specified functions are fulfilled Correctness: describes the degree to which the functions implemented correspond to the specification
User friendliness	Learnability: describes the requirements the product must meet, describes the applications a new user needs to operate the system Manageability: which product requirements are set with regard to convenience in the menu or mask prompts
Time behavior	Describes the properties which define the speed of the product. Some examples of speed features are: • initialization times at login • response times in the event of queries • reaction times/response times • consumption behavior
Maintainability	Describes requirements with regard to, e.g. • interface simplicity • module independence • update capability and cost of reloading when faults occur • inline documentation
Portability	e.g. portability, adaptability
Reusability	Requirements in terms of products or product parts that provide for subsequent reusability

Appendix 2

Audit Management Architecture

High Level Design

V. 1.0

Document management

Document history

Version	Status	Date	Responsible	Reason for change
0.1	in progress	2008–04–02	Carlos Sousa	New creation

Change authorization

Name	Department	Location

Purpose of the document

This is a high level software architecture document for Audit Management application that contains the result of the design phase. It identifies and explains architectural elements and important design decisions. This document will serve the needs of stake holders to understand the system concepts and give a brief summary of the technologies involved in application architecture.

Authoring tools

Word 8.0

Distribution:

Name (in alphabetical order)	**Department**	**Location**
Adélio Fernandes	QPT IT PA DQ	Porto
Alberto Pereira	QPT IT PA AS	Porto
Carlos Sousa	QPT IT PA AS	Porto

Contents

1 Overview

This document aims describing and justifying the new architecture and technologies for the Audit Management application. As Audit Management is an application that already exists it is necessary to define the migration roadmap from old architecture to the new architecture. This roadmap will be detailed in this document.

The document provides a high-level description of the goals of the architecture, the use cases supported by the system and architectural styles and components that have been selected to best achieve the use cases.

2 Integration Test Cases

The test cases are a tool for validating the implementation of the specified requirements by the project's clients. Usually, the test cases are detailed in an Excel file and the testers have to execute them and insert the results of the execution in the Excel file in the line correspondent to the test performed. The results may be of three types: ok, not ok and change request. The change request response signifies that the test will be validated if a small change is done.

The purpose of the test cases is to guide the testers of the application, therefore, in this table are described the test cases of the Audit Management application.

Test N.º	**Functional Area**	**Role**				**Short Description**	**Expected result**
		Auditor	Auditee	Admin	Read-only		
1	User Access	Auditor	Auditee	Admin	Read-only	Login in the application	User is able to login into application.
2	Help menu	Auditor	Auditee	Admin	Read-only	User manual download	There are some user manuals for the application that may be downloaded by all users.
3		Auditor	Auditee	Admin	Read-only	View Audit Management details	View details about Audit Management application like version, copyright and so on.
4	Information view	Auditor	Auditee	Admin	Read-only	Search audit	Search audits applying filters in some filters
5		Auditor	Auditee	Admin	Read-only	View audit details	View all the information about one specific audit.
6		Auditor	Auditee	Admin	Read-only	View personal information	View details of logged on user like email, username or domain.
7	Audit creation	Auditor		Admin		Create audit	Request of a new audit that will trigger all the processes needed to audit achievement. It's needed to introduce some information about the audit that will be performed.
8				Admin		Define schedule	Define date and time when audit will be performed. In process audits and functional audits this schedule may be negotiated with the auditee.
9			Auditee	Admin		Accept schedule proposal from auditee	After a first schedule proposal, auditee can accept or reject the proposal. If he reject, he can itself propose a new schedule that can be accepted or not by the auditor.
10			Auditee	Admin		Reject schedule proposal from auditee	After a first schedule proposal, auditee can accept or reject the proposal. If he reject, he can itself propose a new schedule that can be accepted or not by the auditor. If the auditor rejects the auditee proposal he has to define a new definitive schedule for the audit.

11			Auditee	Admin		Define new schedule	If the auditor rejects auditee proposal for audit schedule, he has to define a new audit schedule that will not be negotiated.
12	Actions verification	Auditor		Admin		Verify action	In consequence to an audit, there are some actions that must be performed to correct deviations detected. After action completion, auditor has to verify if the action was properly done and if it corrects the deviation. Auditor's opinion about the action should be introduced in Audit Management application.
13		Auditor		Admin		Validate action	After verifying action results, auditor may validate or reject the action. If action is done before due date and contributes to correct the deviation, auditor should validate action. To validate action, auditor has to update action status to "Done".
14		Auditor		Admin		Reject action	After verifying action results, auditor may validate or reject the action. If action due date is overdue or action don't contribute to correct the deviation, auditor should reject action. To reject action, auditor has to update action status to "Not Done".
15		Auditor		Admin		Authorize/reject changes to action closed	After auditor verification action may be closed what implies that it can't be edited. This use cases aims to introduce the possibility to edit action details even if action is closed. Changes are performed by auditee that has to request an authorization from auditor which can authorize or not the changes.
16	Audit report	Auditor		Admin		Add deviation	When auditor is performing an audit he encounters some deviations that must be corrected to maintain quality patterns. Those deviations have to be added to audit page in Audit Management application. This use case includes de insertion of the date when audit was performed and supports the insertion of an attachment to the audit page.
17		Auditor		Admin		Compile preliminary report	After insertion of all deviations detected by the auditor, he has to indicate that all deviations have been added. This task is identified like preliminary report compilation.
18		Auditor		Admin		Compile final report	Preliminary audit report may be discussed with the auditee that may disagree with some points. In this case, auditor should review audit report and compile a new version that will be the final version. If auditee doesn't express his disagree with preliminary report it will be automatically converted to final report.
19		Auditor		Admin		Edit deviation	Deviation attributes may be updated by auditor until final report compilation
20		Auditor		Admin		Close final report	After final report compilation, there is a phase where auditee and action owners perform actions needs to correct deviations. When all actions are done, auditor will verify those actions and, if he thinks that deviations are resolved, he may close final report. After that, no changes can be made to audit report.
21	Statistics reports	Auditor		Admin		Refresh data sheet	Users may refresh data sheets presents in MS Excel files for report types. They may refresh all data sheets or refresh by category.
22		Auditor		Admin		Browse statistics report	There are several statistics reports that can be generated, so, user can customize report that he wants to view.
23		Auditor		Admin		Statistics report search	User may search statistics report indicating the report type and other details about the report like audit category, action owner or report status.
24	Action		Auditee	Admin		Update action	Actually, action owner doesn't interact

	achievement					status	with Audit Management application but, with this use case, it's intended to give him a role in the application. After achieved action, Action owner should update status of actions that he's responsible by a comment insertion and puts action as "Done".
25	Schedule negotiation		Auditee	Admin		Accept schedule	After audit schedule proposal by auditor, if auditee agrees with schedule he may accept auditor proposal.
26			Auditee	Admin		Reject schedule	After audit schedule proposal by auditor, if auditee disagrees with schedule he may reject auditor proposal and has to suggest a new schedule. Audit schedule may only be rejected once.
27	Preliminary report negotiation		Auditee	Admin		Accept/reject preliminary report	Auditee can accept or reject preliminary audit report. This consists in accept or reject all the deviations identified by the auditor which are present in the preliminary report. If auditee rejects preliminary report he shall insert remarks to justify its rejection and indicate a date for a meeting to discuss rejected deviations.
28	Actions		Auditee	Admin		Respond on action	After final audit report compilation, auditee has to respond with actions to correct deviations. Auditee has to define action details, indicate the person which will be responsible for perform action and define the due date for action.
29						Confirm action status	After action's achievement by responsible person, auditee has to confirm that action is done. It's needed auditee confirmation to advertise auditor that action was performed.
30	Administration operations			Admin		Add/remove checklist	Each audit category has a list of items that should be checked. Those checklists may be added or removed from Audit Management application.
31				Admin		CRUD customers	CRUD customers of Qimonda that may request external audits.
32				Admin		CRUD audit shift	CRUD shifts when audits may be performed.
33				Admin		CRUD audit elements	CRUD audit elements associated to audit categories.
34				Admin		CRUD departments	CRUD departments where audits may be performed.
35				Admin		Define department roles	This use case allows defining the person that is the head of the department, the process owner, the final report receivers and a distributor list.
36				Admin		Define report cutoff	Define date for report cutoff.
37				Admin		CRUD user groups	A user group is a set of permissions that can be attributed to Audit Management users. It will be possible to create news groups, update and delete existing groups.
38				Admin		Add/remove users from group	Users of Audit Management application may belong to several user groups, so, this use case aims to add or remove a user to some group.
39				Admin		View access log	User login and logout details between some dates will be displayed in access log.
40				Admin		Update help version	There should be a log of versions of help menu, so, when a change is performed in help menu it has to be registered.

3 Architecture

The analysis of the Audit Management architecture was lead by the recommendations of ISO 9126 for software quality and some specific needs due to the context where application will be used.

The most important factors considered in architecture definition are the following:

- **Functionality** – with the defined architecture it is possible to implement all the requirements specified.
- **Reliability** – the application will depend on the fewest possible external systems to reduce the probability of failures.
- **Efficiency** – this factor is particularly important in the disk space taken by Excel reports and in the performance of reports generation.
- **Maintainability** – this property was enhanced by application users due to small differences existing in audit flows in some Qimonda sites that lead to some changes in application source code. Architecture should, also, be modular to make easier the implementation of any new functionality like, for example, the generation of a new kind of report.
- **Portability** – application will be installed in some Qimonda sites and could be frequently transferred from one environment to another, then, it's important to be easy to install and configure.
- **Usability** – user interface of the application should be maintained by order of end users and, thus, it is avoided new learning of application interface.

This architecture was defined considering the reengineering of the Audit Management application, then, it is indispensable to support, at least, all the features previously supported with the same or better performance. In this architecture definition there have been considered also some specific requirements proposed by the application users like modularity and flexibility of source code.

At last, there are two other important factors that governed the architecture definition: reuse the maximum of existing components of source code and follow some design patterns that describe best practices for some architecture mechanisms.

3.1 Logical View

Architecture is defined in three main layers according to the pattern Model-View-Controller. Each component groups similar functionalities and provides them to other components.

The business layer is divided in three secondary layers to separate the code behind of web pages, some components useful to business logic of application and components responsible for accessing database.

The figure below presents a logical view of the application architecture and next are described all the components and technologies used in their development.

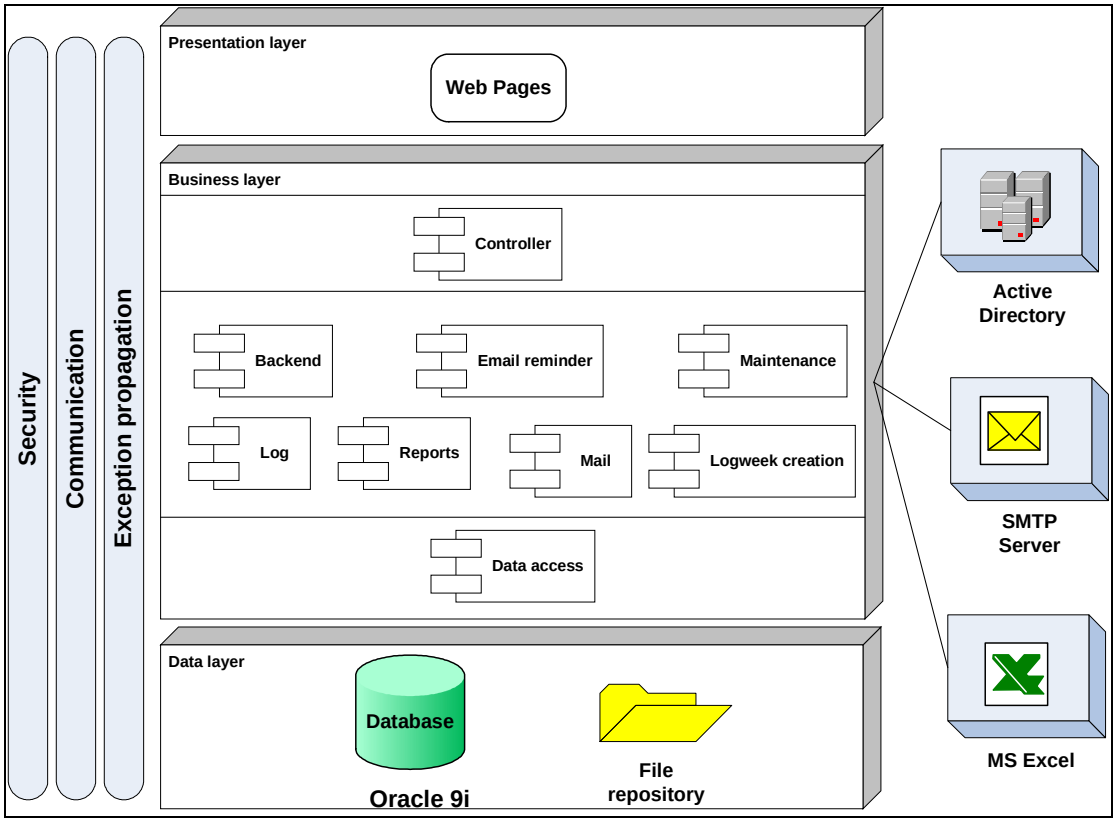


Figure 3 - Logical architecture

- **Web pages:** the user interface of the application consists of a web interface, so, it is accessible across a web browser. The web pages are developed in the .NET 2.0 framework using ASP .NET, HTML, CSS, and JavaScript languages.
- **Controller:** component that contains the code behind the web pages and manages all the business logic of the application, that is, it acts as a coordinator of all the modules of the architecture. It is developed in C# language.
- **Reports:** component that contains functions to manage the generation of reports including their creation, preview, update and deletion. It is developed in C# language and interacts with Microsoft Excel because the reports are generated into Excel files.
- **Maintenance:** component responsible for the management and configuration of maintenance operations. These operations include administration tasks and backend tasks like the deletion of temporary files, the conversion of some pending reports to final reports and the update of the reports data sheet. In the old Audit Management application these operations were executed in a separated executable file, so, when user wanted to perform the maintenance operations he had to run this file. In the new application, these operations are available through the web interface.
- **Backend:** this component is external of the application's core because it is an assembly file (.dll) responsible for performing the backend operations. These operations may be executed occasionally through the Audit Management application or they may be executed periodically with the support of the Windows Task Scheduler.

- **Backend caller:** it is an executable file that will work as an intermediary between Windows Task Scheduler and Backend.dll, that is, the Windows Task Scheduler will be programmed to run this file periodically and this file will invoke functions present in the Backend.dll file. This file is needed because the Windows Task Scheduler only works with executable files, so, it can not invoke the Backend.dll.
- **Log:** component that contains all the functions necessary to log the main operations done in the application.
- **Mail:** component responsible for the sending of emails and the definition of their contents. In this application there are several emails that have to be sent with different contents and to different recipients according to the type of the email. The generation and the sending of those emails are made inside this component.
- **Email Reminder:** In the logical view of the architecture, this component belongs to the business logic layer but it will be installed in database. It is an Oracle database job responsible to send emails periodically depending on some conditions present in database tables. For example, an email should be sent daily to an auditee if the deadline of the execution of its actions is over. The decision to implement this component in the database is due to questions of performance because all the data necessary to decide if an email has to be sent or not is present in database, so, this functionality is entirely executed in the database. This component is developed with the PL/SQL language in an Oracle 9i database.
- **Logweek Creation:** As the Email Remainder component, it is also physically present in the database because it consists of a database job that has to run yearly. This job fills a database table that contains all the weeks of the fiscal calendar of Qimonda. It is developed in PL/SQL language.
- **Data Access:** component that contains all the functions responsible for all the accesses to the database.
- **Database:** component responsible for the storage of the persistent data of the Audit Management application.

3.2 Physical View

In the physical view are visible all the hardware components needed to implement the application's architecture. In images below are presented the actual physical architecture of Audit Management and the physical architecture after the application reengineering.

The unique visible difference is the replacement of the YODA Server by Active Directory Server but this last will be rarely used. This server is used to support user authentication when user that wish to log on the application isn't the same that is logged on the user workstation what means it's predictable that it will not be often used.

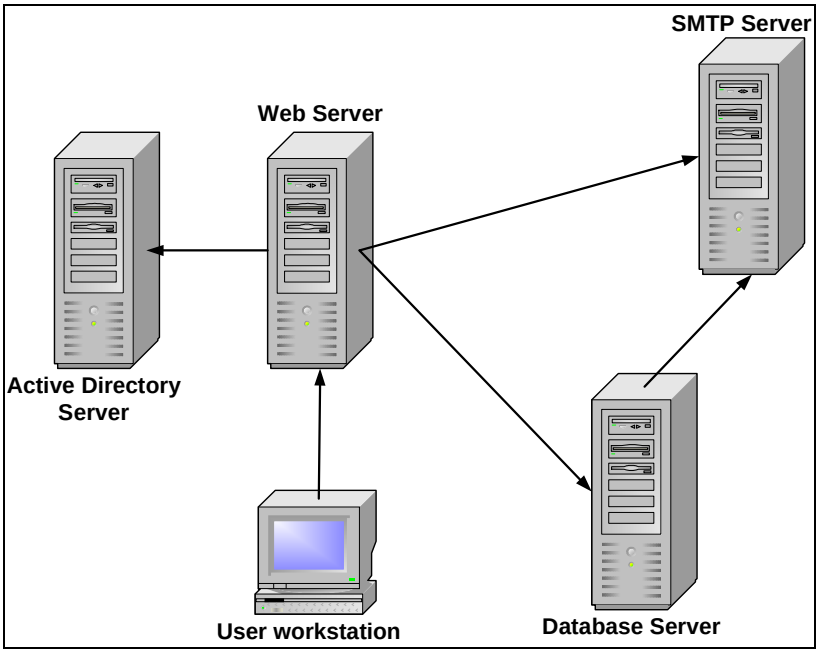


Figure 4 - New physical architecture

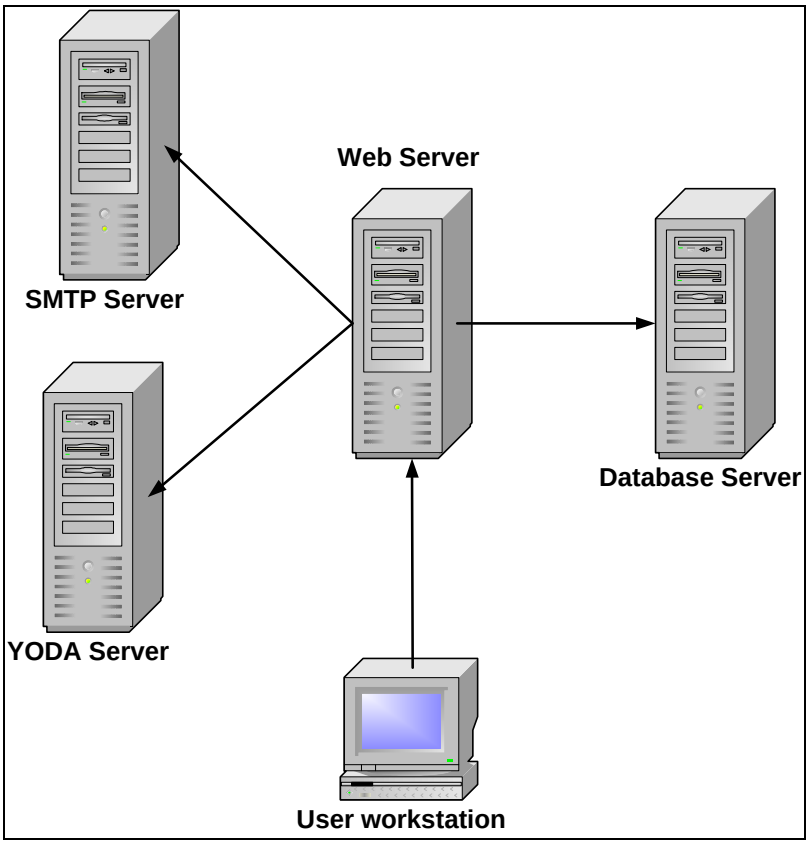


Figure 5 - Old physical architecture

3.3 Deployment View

The deployment view provides a detailed view of the way how components will be deployed across the system infrastructure. It is possible to see where and how components will be deployed and relationships among them.

The figure below shows a deployment view of the application components described in section 6.1 and contains, also, other components that already exist and are detailed after the figure.

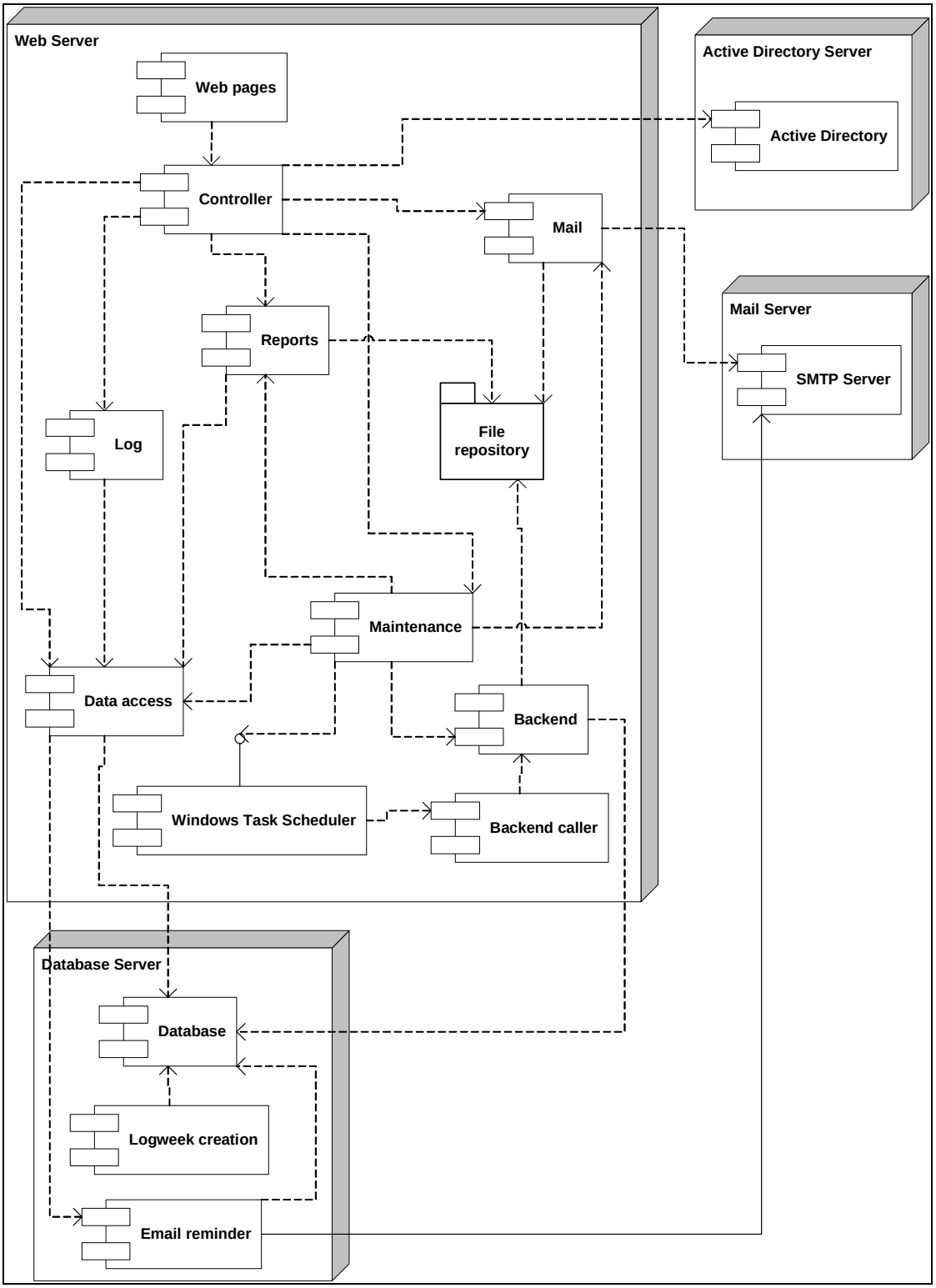


Figure 6 - Deployment diagram

- **File repository:** it is a folder hierarchy located in the web server that aims to store auxiliary files needed by the audits including templates, audit reports, audit notices or audit checklists. These files are Microsoft Excel files.

- **Windows Task Scheduler:** component of the operating system needed to schedule backend operations for Audit Management application. The schedule of this task could be done directly in the Control Panel of operating system or in the Audit Management application.
- **Active Directory:** system where is stored the information about all the users that can log on the Qimonda network.
- **STMP Server:** Server used for email sending.

During the architecture definition, various architectures were considered and all of them were able to support all the requirements of Audit Management but, after analysis of important details, the architecture presented above was selected. However, it is important to refer one alternative to this one that have some benefits but wasn't selected because one of the main goals of architecture is to have a standalone application and depend on the minimum of external systems.

The figure below presents a deployment view for a possible architecture using YODA middleware and COM+ components.

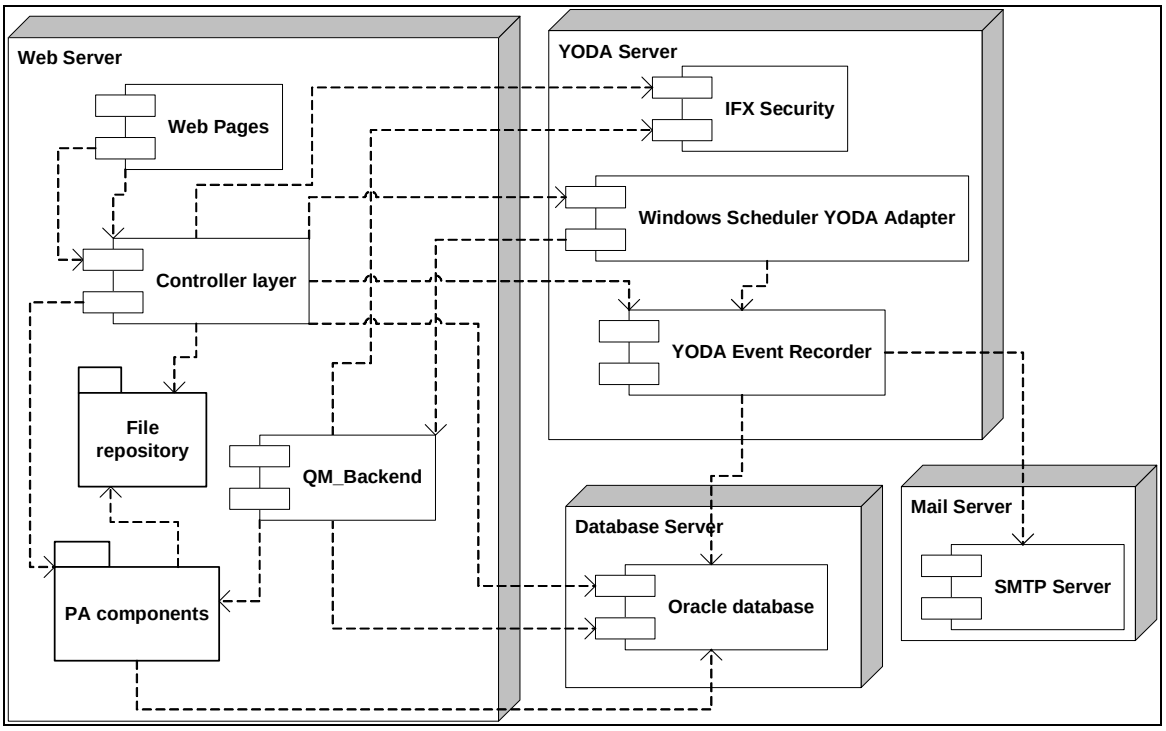


Figure 7 - Deployment diagram with YODA

This solution uses some applications provided by YODA middleware that reduce development effort:

- **IFX Security:** management of application users, roles and authentication.
- **Windows Scheduler YODA Adapter:** application that interacts with Windows Scheduler and provides an easy way to configure periodically tasks. In this case it would be useful to schedule backend operations executed by the COM+ components and email reminders.

- **YODA Event Recorder:** application that records data in four different targets (database, email, file and Windows Event Log). It would be helpful to do application logging and to send emails.

4 Main design decisions

4.1 .NET technology

The adoption of the .NET technologies was the first project requirement because it consists on a migration project from obsoletes technologies to the new .NET standards. Usage of the .NET framework improves application modularity, flexibility and performance and contributes significantly to a well-designed application.

4.2 Pattern Model-View-Controller

This pattern is constantly recommended as a best practice in software design and, in this case, is indispensable to give to application the flexibility and extensibility required by customers. It consists in separate data needed by application, business logic code and user interface code making, thus, code reusable.

In .NET architecture, we can find a correspondence for these concepts:

- View – implemented in files with .aspx extension that contains various languages for web development like ASP .NET, HTML, CSS, JavaScript, etc…
- Controller – contains the code-behind of web pages and is written in C# language and manages the events supported by .NET framework.
- Model – framework provides mechanisms for database connections and encapsulates data within objects that can be used by business logic code.

4.3 Relational model

The design of database will be the same that is actually used and the layers above data layer should interact with it. When After integration of the three layers an analysis of relational model should be done to verify if it's the better model for database or if it has to be changed to another model according to performance and design of database.

4.4 Configuration

Audit Management application has a lot of details that should be configurable. The way selected to maintain configurations needed is with configuration files (.ini or .config) that confers legibility and facility to make changes. Application configurations should consider database connection, SMTP connection, Active Directory connection and schedule of backend operations. Another configuration should be done for database jobs but these are achieved directly in Oracle database.

4.5 Data access

The Microsoft .NET Framework provides a class library, ADO .NET, that can be used to access data and data services. In this project, ADO .NET functions will be used to establish connection with Oracle 9i database and manage the data stored in it.

4.6 Queries or stored procedures

This question is pertinent in the scope of database migration because if database structure will be changed then implies changes in almost all the queries present in business logic layer. If queries are replaced by stored procedures it is not necessary to perform any change to business logic code when database is changed. The only changes that have to be done are in data layer but replacement of queries by stored procedures implies a development effort bigger than make changes to queries after database restructuration.

4.7 COM+

The COM+ technology is used in actual Audit Management application to generate reports and to send emails but COM+ is now deprecated in favor of .NET framework which provides rapid development tools. To make this application less dependent of other technologies, COM+ components will be replaced by C# code integrated in business logic layer. However, this transition will be done carefully and progressively due to complexity of code responsible for reports generation.

4.8 Authentication

Currently, authentication in Audit Management is performed in two steps. First, it's made YODA authentication, that is, an user can log on Audit Management if he's present in IFX Security application. After that, application verifies if user is present in Audit Management database and check user data. This mechanism implies that users must be imported from Active Directory database to YODA and, following, to application database.

After the application reengineering, the authentication will be performed in a different way. To simplify this process and to remove YODA dependence, users will be imported from Active Directory to application database only. The matter here is that only authentication in application database is not enough because users can be removed from Active Directory database and still be present in application database, so, it's need to confirm if user is present in both databases. This problem can be partially solved with the introduction of a new useful feature to application: auto-login. This feature makes user authentication automatic and transparent for user and avoids him introduction of his username and password that are already introduced when user made his authentication in personal computer.

New authentication mechanism will take user credentials from personal computer and will authenticate him immediately in Audit Management application and, like authentication in personal computer is done with verification in Active Directory database, is granted that user has not been removed from Active Directory. If user that wants to log on Audit Management application is not the same that is logged on personal computer, application provide the possibility to make authentication with another user. In this case, two checks must be done: one in application database and another one in Active Directory database.

Users that want to authenticate in Audit Management have to be in application database, so, to avoid put there directly, a feature will be added in application interface to insert new users in database.

4.9 Logweek creation

The fiscal calendar of Qimonda is different than regular calendar and Audit Management needs to interact with fiscal calendar. There is a table in database that makes correspondence between these calendars. For each fiscal week (also called logweek) are indicated the days of start and end of week in regular calendar.

To fill this database table, it will be implemented a database job that will run once a year without intervention of user and will update logweeks details for a year.

4.10 Backend operations

Currently, backend operations are performed outside application by an executable file whose schedule of execution may be defined in Windows Task Scheduler or this file may run only when user decide to execute it.

After application reengineering, these operations can also be performed across Audit Management interface but to allow this executable file will be replaced by an assembly file (.dll) and an executable file (.exe) for security reasons. Functions to perform backend operations will be contained in assembly file (component backend in deployment view) and will be invoked by executable file (component backend caller in deployment view) or by Audit Management application. Executable file exists to allow backend operations schedule. Windows Task Scheduler may be configured to invoke executable file that, in turn, will call functions contained in assembly file and, thus, perform backend operations.

This division in two files is done to avoid give permissions to run executable files in web server across web interface.

The backend operations actually performed by Audit Management and the way how they will be implemented are detailed in the next sections.

4.10.1 Refresh users

This operation consists in import users from YODA to application database but, like YODA will be removed from application architecture and the way of importing users will be different, this operation will not be needed.

4.10.2 Delete temporary files

The temporary files present in file repository that aren't need any more will be deleted with execution of this operation. Functions to do this are included in assembly file.

4.10.3 Convert final report

The reports that aren't closed and that limit date to report closure is previous to date when backend operations are performed must be automatically closed. Functions to do this are included in assembly file.

4.10.4 Refresh data sheet

This operation consists in update data of some Excel files according to values present in database. Functions to do this are included in assembly file.

4.11 Logging

Currently, it's only done logging of user login and logout but application reengineering will introduce a more detail log of operations including, for example, audit requests and audit report closure. A class to treat logging will be implemented and will provide methods to other classes.

Logging of operations is useful to debug an eventual error in application workflow but degrade performance, so, it is necessary to log only the most important operations.

4.12 File repository

The file repository will maintain the same structure and hierarchy and will be present in web server.

4.13 Reports generation

The code for reports generation is actually contained in COM+ components and is complex and extensive but the goal of this architecture definition is to replace these components by some classes written in C# that will be included in application core.

4.14 Email reminder

Currently, this mechanism involves Windows Task Scheduler to schedule email sending, a COM+ component to send an email and a database where is data needed to compose email. To improve performance and application independence, email reminder will be implemented by a database job. This job takes the data needed in the database, compose the email and send it every day until to reach conditions to stop.

5 Migration Plan

The migration of Audit Management is a complex process due to the application dimension and to the wide distribution of its components (COM+, YODA, database, email reminder, etc…). To achieve a successful migration it's indispensable to do it carefully and progressively and grant that application is always functional.

The migration approach will start in top logical layer to lower logical layer and is divided in four main phases that group similar components or components than can't work separately. Each phase aims to replace a set of application components and reuse others and, at the end of the phase, application must work properly for all the features.

The migration is not only source code migration because there are some mechanisms that will be implemented in a different way and have to be included in migration process. Moreover, evolution of technology introduces some features and components (dataset, datareader, ADO .NET, .NET events, etc…) that improve performance, security and flexibility and should be used in application development in detriment of old solutions.

The figures below present a deployment view of actual application architecture and will help to identify all the steps of migration process.

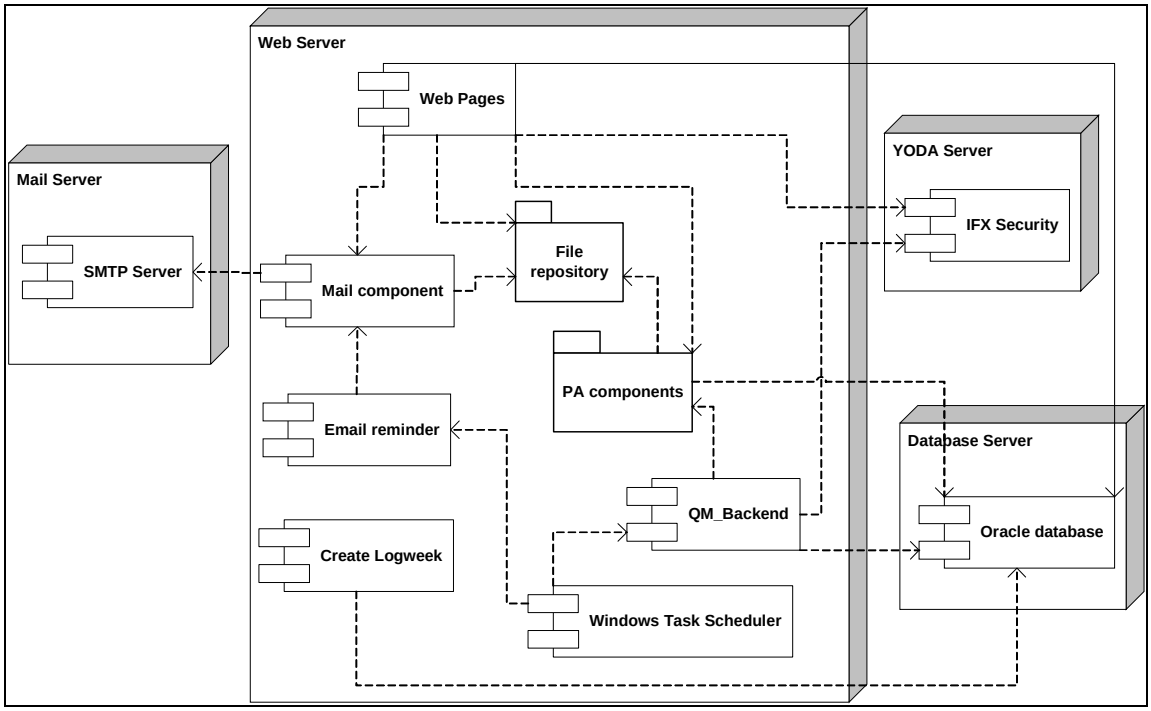


Figure 8 - deployment diagram of actual application

It is possible to see that the application core is in the Web Pages component. The figure below shows the content of the PA Components folder.

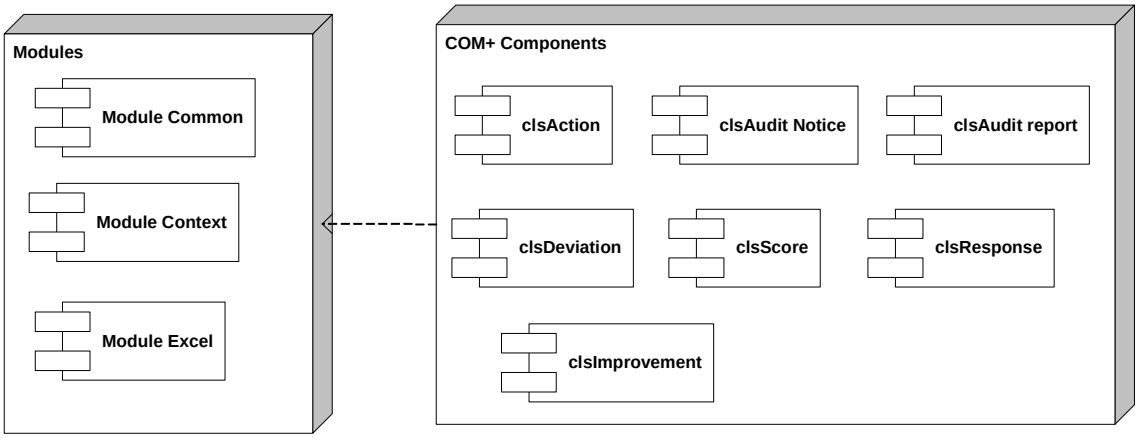


Figure 9 - COM+ modules and classes

The next sections describe the migration phases in the order they happen.

5.1 Phase 1 – Core application migration

The first migration step consists on migrating the user interface but it's impossible to migrate only the code for presentation layer due to organization of actual source code files. Actually, presentation code and business logic code are mixed in the same file, so, some components of middle layer have to be migrated in this phase to maintain application functional. Thus, all the content of Web Pages component visible in deployment view of actual application will be replaced

by a set of components that implement the application core and external components will be maintained.

The components that appear after this phase are:

- Web pages
- Controller
- Mail
- Data access
- Log

The authentication mechanism is also updated in this phase leading to replacement of YODA system by Active Directory.

To achieve this step, files with .asp extension should be replaced by files with .aspx extension and ASP code and Visual Basic Script will be replaced by ASP .NET and C# code.

After this phase the deployment view of application architecture is in figure below.

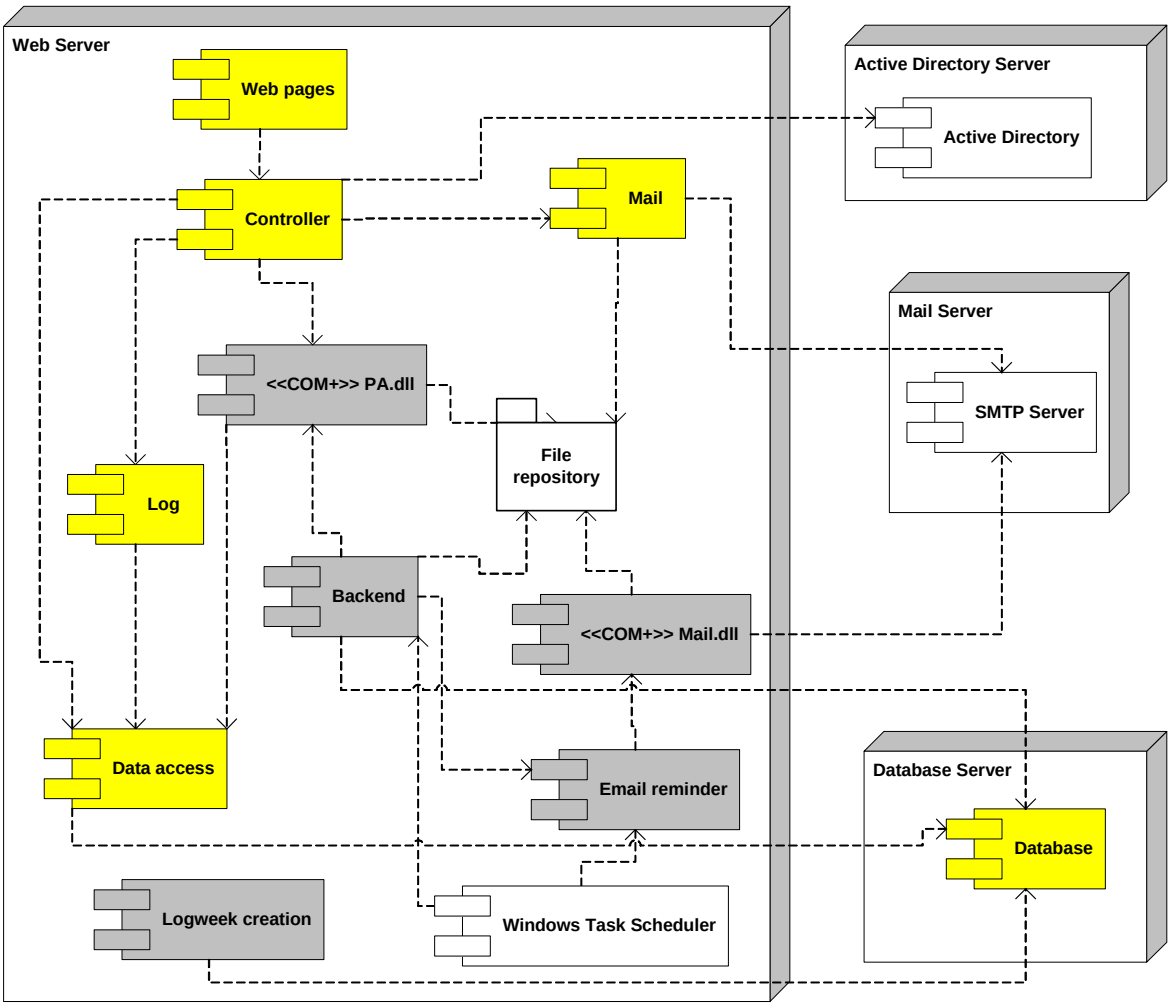


Figure 10 - intermediary deployment diagram

5.2 Phase 2 – External features migration

The second phase of the application migration concerns components outside the application core and new components to be added to this architecture.

At this step application is logically divided in three main layers, so, it's necessary to integrate other components in this structure.

The components that will be developed in this phase are:

- Maintenance
- Backend
- Backend caller
- Email reminder
- Logweek creation

The Maintenance component implements a new feature that doesn't exist currently in the application, this is, it provides execution of backend operations across user interface and will be integrated in the core of application.

Backend and Backend caller are already described in a previous chapter and both will be implemented using C# language.

Email reminder will replace actual executable file and will be implemented as database job in PL/SQL language.

Logweek creation is a component that will also replace an executable file to fills database with Qimonda fiscal weeks and it will be implemented as a database job. The benefits brought with this change are the avoidance of user intervention to execute this operation because it's scheduled in database and, thus, this operation is not distributed by an executable file and database.

5.3 Phase 3 – Reports generation analysis and migration

At this point all application components are migrated excepted COM+ components responsible for reports generation but it's not granted that these components have to be migrated.

In this phase, it will be done an analysis to reports generation with the aim to identify the better way to make reports more flexible and their generation more efficient. One alternative to technology actually used to support reports (MS Excel) is Business Objects technology that should be considered in this analysis. Other technologies can also be studied to verify if they have some benefits relatively to Microsoft Excel.

If the analysis conclusion indicates that the way actually used to generate reports is the most appropriate to that, then, COM+ components written in Visual Basic 6 language will be migrated to C# and integrated with other components of application. The result of this migration represents the Reports component present in deployment view of architecture.

5.4 Phase 4 – Database analysis and migration

At this point the presentation layer and the business logic layer are migrated and are supported by the same database. It's not mandatory that database has to been migrated, so, an analysis should be performed to indentify if the benefits brought by database migration are enough to justify a restructuration of database.

6 Document/literature references

- ISO 9126 – http://en.wikipedia.org/wiki/ISO_9126
- ADO .NET – http://en.wikipedia.org/wiki/ADO_.NET
- COM+ – http://en.wikipedia.org/wiki/COM%2B#COM.2B
- Active Directory - http://en.wikipedia.org/wiki/Active_directory